Step-by-Step: Quantum Walk Implementations and Visualizations

by

Addie Jordon
M.Sc., University of Victoria, 2023

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Step-by-Step: Quantum Walk Implementations and Visualizations

by

Addie Jordon
M.Sc., University of Victoria, 2023

Supervisory Committee

Dr. Ulrike Stege, Supervisor
(Department of Computer Science)

Dr. Hausi Müller, Departmental Member
(Department of Computer Science)

Dr. Irina Paci, Outside Member
(Department of Chemistry)

Supervisory Committee

Dr. Ulrike Stege, Supervisor
(Department of Computer Science)

Dr. Hausi Müller, Departmental Member
(Department of Computer Science)

Dr. Irina Paci, Outside Member
(Department of Chemistry)

Abstract—Quantum walks (QWs) are the quantum analogue to classical random walks. I provide background and motivation for quantum walk algorithms and their applications. I present new visualizations for one-, two-, and three-dimensional quantum walks and explain how the visualizations can help teach quantum concepts such as superposition and interference. I contribute the Quantum Walk Visualization (QWalkVis) application for visualizing quantum walks. Users can select the dimensions, number of states, and number of steps in the walk, and generate probabilistic plots on-the-fly. Users are able to view the probabilities for each step of the walk one by one. QWalkVis aims to aid students in learning about quantum walks and foundational quantum concepts through interactive exploration. I present use cases of QWalkVis for both education and research purposes. I also propose a new application of quantum walks for creating noise distributions. Noise distributions are essential for noise sampling used in differential privacy, and help keep query data private.

Table of Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Ulrike Stege, my supervisor, for her constant support and mentorship throughout my Bachelor's and Master's, and for introducing me to quantum computing in the first place! Thank you for passing along the quantum fever; I have had so much fun studying this topic.

all Rigi group members, Austin Hawkins-Seagram, José Ossorio, and Samantha Norrie, for helping me with my research in many ways, whether it be an off-handed conversation, working together on the Quantum Bootcamp, sending me papers that you came across that used quantum walks, or helping me build the Quantum Walks Visualization application.

NSERC CREATE Program, MITACS, QAI, and Multiverse Computing, for funding me throughout my research journey.

Jan Jordon and Lynne Jordon, for helping me proofread and for constantly asking me, 'Have you started writing your thesis yet?'

and Ammar Elnakady, for listening to me run through countless presentations and educational material over the years and reminding me that everything is going to be ok.

Chapter 1

Introduction

Early on in my quantum computing journey, I came across quantum walks. Quantum walks immediately captivated my attention as they possess the quintessential parts of quantum theory in a single, relatively simple algorithm. Briefly summarized, quantum walks feature a walker moving around a given structure, for example walking along a line or a grid. The walker is also equipped with a coin that dictates the direction the walker will move when taking their next step. For example, a walker on a horizontal line will flip a two-sided coin to determine whether to take a step right or left. The walker continues to flip the coin and take a step iteratively until some stopping criteria are met.

Although that description may not initially sound all that exciting, what makes quantum walks so interesting, is particularly the quantum behaviour of the walk. For example, a quantum walk on a line allows for the coin to 'land' with both sides facing up, and the walker then takes a step both left and right at the same time. This phenomenon is called *superposition* and is one of the key concepts of quantum physics. Quantum walks also demonstrate *interference*, another key concept which is described in Chapter 2.

Additionally, quantum walks concisely illustrate the issue of *measurement* in quantum computing. Namely, quantum systems are unobservable while an experiment is being carried out upon them. Although quantum particles can exist in superpositions of multiple states (e.g., a quantum walker can exist in two locations at the same time), when *observed* the particle must 'choose' exactly one discrete state to be in (e.g., the quantum walker must 'choose' only one of the two locations to be in). Such a choice is in actuality not a choice at all, and which location the walker will be in when observed is drawn from a probability distribution and ultimately truly random.

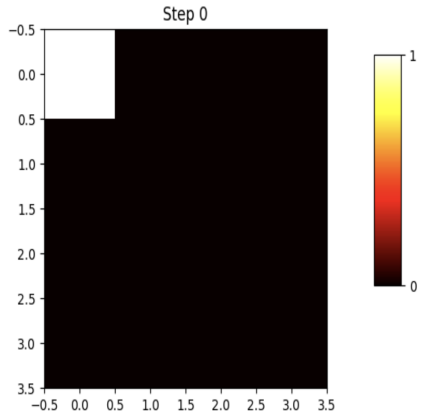
In terms of quantum walks, a quantum walker may take many steps and end up in multiple positions, but upon measurement only one of those positions will be observed with some probability. Thus, often when utilizing quantum walk algorithms, one aims to increase the probability of measuring a desired position.

After my continued study of quantum walks, I believe quantum walk algorithms are not only fascinating, but have the potential to make excellent educational material. Quantum walks can allow one to introduce paramount quantum concepts such as superposition, interference, and measurement in a concise algorithm. The concept of a walker tossing a coin and taking steps in a position space is not only universally simple to explain but also easy to visualize. As a result, the main goal for this thesis was to create educational materials centered around quantum walks to aid in the teaching of foundational quantum concepts. These materials are interactive, require little prerequisites, and inspire potential students to pursue learning more about quantum computing; essentially, the primary goal of these materials is to act as a 'foot in the door' for learning about quantum.

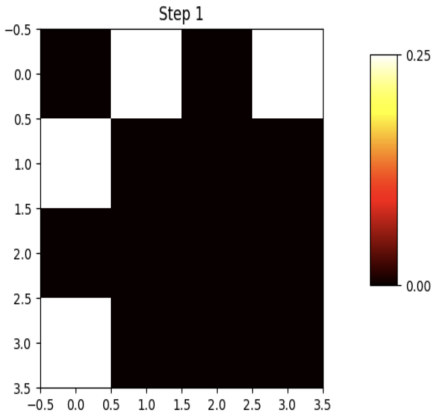
To meet that goal, I created the Quantum Walks Visualization (QWalkVis) application, which is the main contribution of this thesis. QWalkVis is a single page website application that creates novel visualizations of quantum walks on-the-fly. It allows the user to enter parameters for a quantum walk such as how many steps the walker should take, and whether the walk should take place on a line, grid, or cube. After entering the parameters, plots of the walker's probability distribution are made, and the user can examine the probability distributions at each step of the walk (cf. Fig. 1.1).

QWalkVis creates visualizations that are different from existing quantum walk visualizations available on the internet. Other visualizers typically create a line graph of the final probability distribution after all steps in the walk are taken [98, 8, 52, 83, 6]. Although it is possible to create such visualizations for each individual step of the walk, it is difficult to create line graphs for two- or three-dimensional walks. Comparatively, QWalkVis creates visualizations for one-, two-, and three-dimensional walks. QWalkVis also creates visualizations that are 'position-centric'; the structure that the walker is moving on is drawn instead of only the charted probabilities. In order to show the probabilities, each position is coloured according to a colourbar (cf. Fig. 1.2).

Because QWalkVis is a web application, users require no programming knowledge to explore foundational quantum concepts. However, those who are interested in the



(a) The initial position of the walker.



(b) The position of the walker after taking one step using the Hadamard coin.

Figure 1.1: The walker begins at a single position with 100% probability in (a). After taking a step in all four directions, the walker is in four positions each with 25% probability in (b). The colour bar maps the probabilities to a colour gradient.

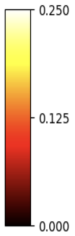


Figure 1.2: The colour of each position denotes the probability of the walker being at that position upon measurement. The colour bar maps the probabilities to a colour gradient.

programmatic implementation of quantum walks can view the code on GitHub[1] or the review pseudocode in Chapter 3. The quantum walk algorithm implementation presented in this thesis differs from the implementations found in literature, as discussed in Chapter 3. Other implementations are usually hardware based and require specialized equipment to recreate. Those that are not hardware based are quite complex and not well suited to beginners of quantum computing. The implementation offered in this thesis and used in QWalkVis is a new simplified and relatively generic quantum walk algorithm that only requires a quantum universal gate set; that is, it

[1]https://github.com/addie43110/qwalk-visualizations

is completely hardware agnostic.

QWalkVis can be used both as an educational tool or for researching quantum walks. For those interested in teaching introductory level quantum computing using quantum walks and QWalkVis, a sample lecture and educational suggestions are provided in Chapter 5.

Sometimes it is helpful to see applications of quantum walks for students to better understand their functionality. A new application for quantum walks as a subroutine is suggested in Chapter 6, which advocates the use of quantum walks in differential privacy. Not only can this application help increase privacy in datasets, but it also serves as an example of the vast potential of quantum walks.

1.1 Thesis Outline

Chapter 2 introduces terminology, background for quantum computing and quantum walks, and quantum walk applications. Chapter 3 discusses the design and implementation of quantum walks. Chapter 4 describes existing quantum walk visualizations and QWalkVis, with use cases for QWalkVis and additional educational material, such as lecture slides and written notes, covered in Chapter 5. Chapter 6 gives an example of using quantum walks for noise sampling in privacy settings. Because quantum measurement is truly random, and randomness is imperative for privacy, quantum algorithms can be used improve the privacy of datasets. Chapter 7 outlines conclusions, limitations, and future work.

Chapter 2

Background

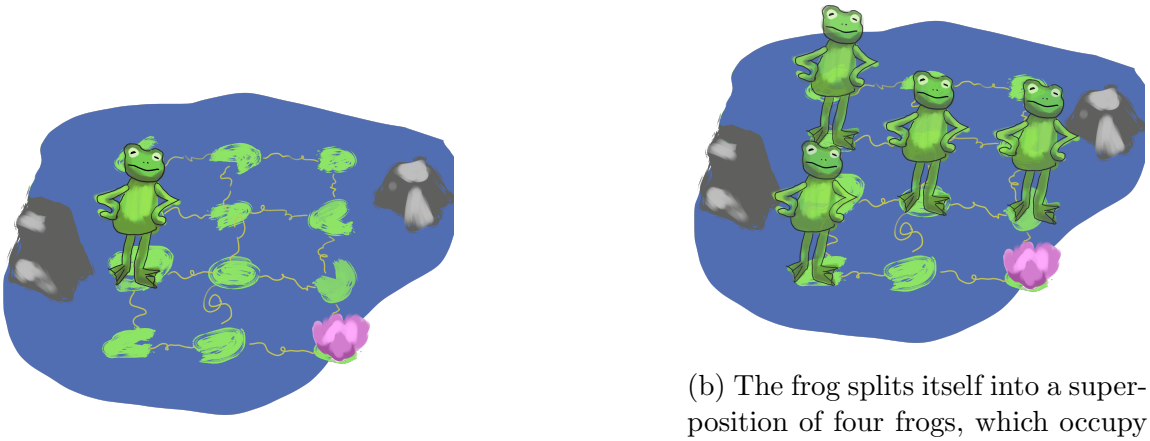
2.1 Prerequisites

The following sections briefly detail necessary terminology for the remainder of this thesis. The reader is assumed to have basic familiarity with linear algebra and graph theory as described in computer science or discrete mathematics.

2.2 Kets, Superposition, and Interference

Classical (day-to-day) computers use *binary digits*, or *bits*, to store information, which can take on the value of either 0 or 1. Quantum computers instead use *quantum bits*, or *qubits*, which can be the states $|0\rangle$, $|1\rangle$, or any *superposition* of the two. The $|0\rangle$ state is analogous to the 0 bit, while the $|1\rangle$ state is analogous to the 1 bit. The $|$ and $\rangle$ symbols are a part of Bra-Ket notation (also called Dirac notation) [32], where $\langle x|$ is called a *bra* and represents a row-vector, and $|x\rangle$ is called a *ket* and represents a column-vector. Thus $|0\rangle$ and $|1\rangle$ are column vectors: $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$. A superposition (cf. Fig. 2.1) of the two refers to any linear combination of the $|0\rangle$ and $|1\rangle$ *basis* states where the combination must also be of unit length. Thus, it is written as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $|\alpha|^2 + |\beta|^2 = 1$. Written as a vector, $|\psi\rangle = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$. Note that the choice of naming the quantum state '$|\psi\rangle$' is an arbitrary albeit common preference.

In a superposition, α and β can take on any complex values and are referred to as *amplitudes*. As a result, when manipulating multiple quantum states with the



(a) A frog begins at a single lilypad.

(b) The frog splits itself into a superposition of four frogs, which occupy four lilypads.

Figure 2.1: A frog crosses a series of lilypads quantumly using superposition.

same basis, the 'α's and 'β's can be added or subtracted, resulting in what is called constructive and destructive interference, respectively. A classic visual demonstration of interference uses two waves travelling toward each other (cf. Fig. 2.2). Another visualization can be made with frogs sitting on adjacent lilypads (cf. Fig. 2.3).

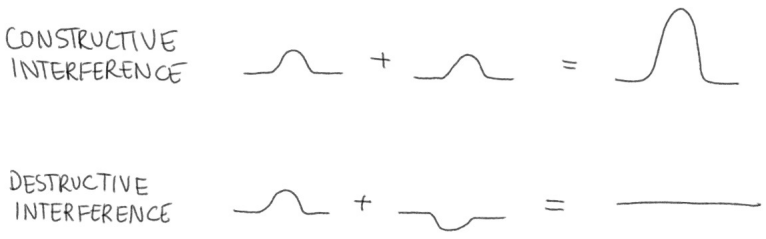


Figure 2.2: Constructive and destructive interference shown by two waves. In constructive interference, the amplitudes of each individual wave sum to create a larger wave; in destructive interference, the amplitudes subtract out to eliminate all waves.

For example, consider $|\psi\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$ and $|\varphi\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$, with $\alpha_1, \alpha_2, \beta_1, \beta_2 > 0$. Then $|\psi\rangle + |\varphi\rangle = (\alpha_1 + \alpha_2)|0\rangle + (\beta_1 + \beta_2)|1\rangle = \begin{bmatrix} \alpha_1 + \alpha_2 \\ \beta_1 + \beta_2 \end{bmatrix}$. Since $|\psi\rangle$ and $|\varphi\rangle$ are added, this is an example of constructive interference. If one was subtracted from the other, it would be an example of destructive interference.

The α and β values of a quantum state $|\psi\rangle$ give the amplitudes of the $|0\rangle$ and $|1\rangle$ terms respectively. The square of these values (i.e., $|\alpha|^2$ and $|\beta|^2$) represent *probabilities*. However, the values of α and β themselves cannot be extracted from a quantum

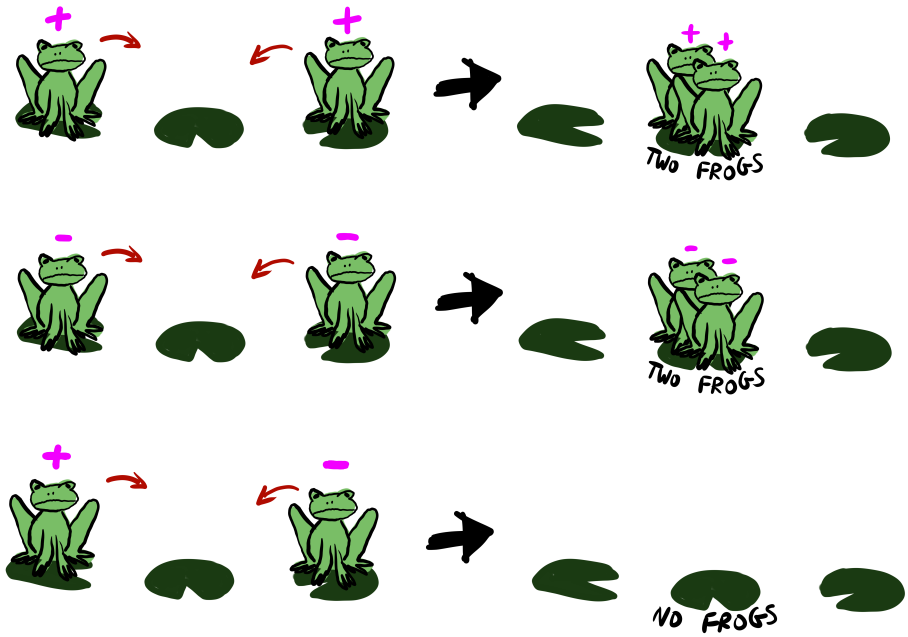


Figure 2.3: Constructive interference shown in the top two lines of frogs, where two positive (+) frogs jump onto the same lilypad or two negative (-) frogs jump onto the same lilypad. In there event of one positive and one negative frog jumping on the same lilypad, they both fall off, showing destructive interference.

state. In order to extract information from a quantum state, one must perform *measurement*, during which the superposition is *collapsed* and a bit is read out with probability proportional to each amplitude squared. For example, when measuring the qubit $|\psi\rangle = \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle$, there is a $(\frac{1}{2})^2 = 0.25$ or 25% chance to read the bit 0 and a $(\frac{\sqrt{3}}{2})^2 = 0.75$ or 75% chance to read the bit 1. In summary, the measuring of a qubit gives a random result in accordance with the qubit's probability distribution, and a qubit **must** be measured in order to extract any information from it.

Multiple qubits states are constructed by applying the tensor product. For example, constructing a two-qubit state from $|q_1\rangle = |0\rangle$ and $|q_2\rangle = |1\rangle$ can be written as $|q_1 q_2\rangle = |0\rangle \otimes |1\rangle$. $|0\rangle \otimes |1\rangle$ is further simplified in Dirac notation to $|01\rangle$. When using an abstract number of qubits, the letter n is often chosen. For example, one might refer to an 'n-qubit circuit' or 'register with n qubits'. To represent n qubits of the same state, the shorthand $|\psi\rangle^{\otimes n}$ is often used, which means that there are n qubits all in the state $|\psi\rangle$.

2.3 Quantum Gates

In order to manipulate quantum states, quantum gates (or *unitaries*) are used. Since quantum states are vectors, it makes sense that quantum gates are matrices. In order to preserve the unit length requirement of quantum state vectors, quantum gates must be *unitary* matrices. Many quantum gates are also *Hermitian* matrices and serve as their own conjugate transpose, meaning that applying such a gate twice in a row results in no change to the state.

Basic gates include the Pauli gates (X, Y, and Z), the Hadamard gate (H), and various controlled gates such as the controlled-NOT (CX) and Toffoli gate. Table 2.1 below summarizes selected basic gates in matrix form; the Toffoli gate is not written since its matrix is large.

Table 2.1: Common quantum gates

Gate (U)	**Matrix**	$U\|0\rangle$	$U\|1\rangle$
X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\|1\rangle$	$\|0\rangle$
Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$i\|1\rangle$	$-i\|0\rangle$
Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\|0\rangle$	$-\|1\rangle$
H	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$\|+\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	$\|-\rangle = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$

The controlled-NOT gate is a two-qubit gate and must be applied to two qubits. One qubit is denoted the *control* and the other is the *target*. In this thesis, the control can be assumed to be the leftmost qubit unless otherwise stated. When the control is set to $|1\rangle$, an X gate (also called a NOT gate) is applied to the target. If the control is set to $|0\rangle$, nothing is applied to the target.

Table 2.2: The controlled-NOT gate

Gate (U)	**Matrix**	$U\|00\rangle$	$U\|01\rangle$	$U\|10\rangle$	$U\|11\rangle$
CX	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\|00\rangle$	$\|01\rangle$	$\|11\rangle$	$\|10\rangle$

The Toffoli gate is a variant of the CX gate with multiple controls. All controls

must be set to $|1\rangle$ in order for the X gate to be applied to a single target. For example, consider a four-qubit Toffoli gate applied to $|q_1q_2q_3q_4\rangle$ where q_1, q_2, and q_3 are controls and q_4 is the target qubit. If $|q_1q_2q_3q_4\rangle = |1110\rangle$, then applying the Toffoli gate will result in q_4 changing from $|0\rangle$ to $|1\rangle$, resulting in the final state $|1111\rangle$. If $|q_1q_2q_3q_4\rangle = |1011\rangle$, then applying the Toffoli gate will result in no change, since q_2 is a control that is not set to $|1\rangle$.

Other controlled gates operate similarly. In most programming languages that suppose quantum circuit construction, custom gates can be controlled as well. However, within all multi-controlled gates, the target qubit(s) cannot include any control qubit(s).

2.4 Famous Quantum Algorithms

There are many quantum algorithms that are widely accepted, but only the ones tangentially related to quantum walks are discussed here.

Grover's search [39] is a quantum algorithm to search an unstructured search space. Consider N distinct elements which are not sorted and in any random order. One of those elements is the target. It would classically take $O(N)$ time in the best case to find the target element, since upon looking at each element one-by-one, the target element could be the very last element you look at. Grover's search achieves a polynomial speedup from the classical algorithm by finding the target in $O(\sqrt{N})$ time. The Qiskit textbook offers more information on Grover's search algorithm and how it works [25].

The quantum phase estimation algorithm [56, 92] is another well-known subroutine with many applications. Phase estimation is able to extract the phase of a quantum state using the quantum fourier transform [26].

2.5 Additional Resources for Getting Started

There are many existing paper and online resources for self-learning quantum computing. A popular textbook, *Quantum Computation and Quantum Information* by Nielsen and Chuang [77], comprehensively covers quantum computation from linear-algebraic notation to famous quantum algorithms. A popular online resource is the Qiskit textbook [25], which teaches quantum concepts through the lens of programming in Python. There exist many more resources such as Xanadu's PennyLane

learning pages [106] and Quantum Codebook [3], Google's Cirq manual [30], and IBM's Quantum Composer tutorial [45]. Advice and resources for learning quantum computing specifically at the university level can also be found [105, 40], as well as courses and workshops aimed specifically for high school students [88, 11]. Books for learning quantum computing include *Quantum Computing for Everyone* [16] and *Dancing with Qubits* [96].

2.6 Oracles

The use of oracles has proven to be beneficial to many areas of classical research. For example, an oracle for pattern matching called the *factor oracle* [4] has applications from machine learning [13], to data compression [62], to genomics [5]. Oracles are also used in software testing [80, 85, 95], complexity theory [17, 43], linear optimization [68], and cryptography [20]. In the most generic case, an oracle is a black-box function (i.e., its implementation and logic are unknown to the user) that returns a yes or no answer when given some input. In practice, oracles are often used to identify if an element is a desired element or *target* in a given search space. For example, consider a domain of positive integers where only a subset, S, of integers are desired, with $S \subset \mathbb{Z}^+$. One could ask an oracle 'Is 6 one of the desired integers?' and the oracle would respond yes if $6 \in S$ and no otherwise.

In that sense, the input to an oracle is an educated guess, and the output is the correctness of the submitted guess. Thus, the oracle knows all answers to the problem before one submits a guess. When sharing the concept of oracles with students, I often encounter the following observation, 'If I must program the oracle, and the oracle already knows all the answers, then I must know all the answers already. Does that not mean the problem must already be solved? Why are we trying to solve it again?'

To some extent, this observation is true. If the goal is to implement the oracle, then one must have a solution to the problem the oracle answers. However, oracles are still of great use for two main reasons. Firstly, sometimes the goal is simply to implement the oracle itself. Secondly, oracles are usually only a stepping stone in a larger algorithm. In the first case, attempting to implement the oracle means attempting to solve a problem whose solutions will help others. The second case naturally inserts itself here; once one has access to the solution of a smaller problem, they can build solutions to progressively larger problems. Often the smaller problem that

the oracle deals with, while important, does not represent the goal of the algorithm that employs the oracle. As a trivial example, consider an oracle which answers the question 'Is it raining?'. The answer of that oracle is essential when evaluating other questions such as 'Should I bring an umbrella with me?' or 'What are the current visibility conditions (for driving)?'.

It can be said that programmers use a variant of the oracle when writing conditional branching. Consider the following pseudocode (cf. Fig. 2.4).

```
1                if (x > 10)
2                    foo()
3                else
4                    bar()
```

Figure 2.4: Common conditional branching in programming can be viewed as a basic oracle.

Here, we compare an integer variable `x` to see if it is larger than `10`. If `x > 10` is `True`, we execute function `foo()`. Otherwise, `bar()` will be executed. Here, the `> 10` part of the condition can be viewed as a white-box oracle. We submit the input `x` and receive an answer of either `True` or `False`.

This example illustrates a common use case of oracles where the user *does not know the exact value of the input*. Perhaps it is important that the value of `x` stay hidden for privacy or, more likely, it simply does not matter what `x` is as long as it is larger than 10.

Another point this example highlights is the fact that oracles often do not compare input against a list of predetermined answers, but rather check that the input meets a set of criteria. A common fallacy is to think the oracle magically, instantaneously knows all the correct answers. For example, a student learning about oracles might think that an oracle that determines if an integer is prime or not must have access to a list of all possible prime numbers. More likely, such an oracle would repeatedly try to divide its input by other integers in order to determine if its input is prime. In terms of the example in the pseudocode, the oracle does not have a list of all integers larger than 10, but instead checks that `x` meets the criteria of being larger than 10.

2.7 Classical Random Walks

As classical random walks provide much of the same structure and terminology used by quantum walks, a brief introduction is included here to familiarize the reader with concepts that will be built upon when discussing quantum walks.

Consider a walker on a connected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. The walker initially stands at vertex $v \in V$. With each step, the walker may move to an adjacent vertex w (with edge $\{v, w\} \in E$). If there are m adjacent vertices to a vertex v, the walker rolls a fair m-sided die to determine which vertex they will step to next. In this scenario, there is a probability of $\frac{1}{m}$ that the walker steps to any single adjacent vertex w.

The edges incident to v may have different weights. Let A be the adjacency matrix of G where A_{ij} represents the weight of edge $\{i, j\}$. In the event where certain edges incident to v are weighted more than others, the probability that the walker steps from v to w becomes $p_{vw} = \frac{A_{vw}}{\sum_{w \in V} A_{vw}}$.

A key concept of random walks is the *hitting time*, which is the expected number of steps required to reach vertex j starting at vertex i. The hitting time can be calculated through the recursive definition

$$h_{ij} = \begin{cases} 1 + \sum_{k \in \mathcal{N}_i} p_{ik} h_{kj} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

where $\mathcal{N}_i$ is the set of all neighbours of vertex i and p_{ik} is the probability of walking along edge $\{i, k\}$ [107].

2.7.1 Applications of Random Walks

The PageRank [78] algorithm, designed for use on the World Wide Web, ranks webpages relative to each other. For example, a webpage receives a high ranking when there exist many other webpages which contain links to it. Since webpages can be represented using vertices and hyperlinks represented by directed edges, the problem domain can be modeled as a directed graph (called the webgraph). The rank of each webpage corresponds to the probability distribution of a random walk on the webgraph [107].

Random walks with reset can be used as a type of randomized search algorithm wherein a random walk is carried out on a search space but with some probability

r the walker may 'reset' (for example, to the initial position) at any step during the walk. Randomized search algorithms often operate using two types of moves: short-range and long-range [69]. The walker searches for the target during short-range moves and jumps to relatively distant sections of the search space during long-range moves after some predetermined time. The 'reset' can be seen as a long-range move that is desirable when the walker has not found the target in their current area of the search space. In this case it intuitively makes more sense for the walker to search a different space than to continue taking short-range steps.

Random walk algorithms can also be used for image segmentation [36]. Given a small number of predefined labels (seeds), the hitting times (or similarly, probabilities) are calculated for a walker starting at any given pixel to reach each one of the initial seeds. Segments are formed based on which seed has the shortest hitting time. The walker is biased so that they will avoid sudden changes in intensity (i.e., borders).

2.8 Quantum Walks

There are two main types of quantum walks: continuous time quantum walks and discrete time quantum walks [98]. The difference between the two lies in the 'step' operation. In discrete time quantum walks, the walker iteratively tosses a coin and takes a step at specific (usually equal) time intervals. Continuous time quantum walks use a Hamiltonion to instead 'evolve' the state over time (i.e., the walker may apply the evolution Hamiltonion at any time). For the remainder of this report, any mention of 'quantum walks' refers to discrete time quantum walks unless otherwise specified.

2.9 Discrete Time Quantum Walks

Like classical random walks, quantum walks can be carried out on a graph $G = (V, E)$. More generally, however, they may be carried out on any Hilbert space $\mathcal{H} = \mathcal{H}^D \otimes \mathcal{H}^P$, where $\mathcal{H}^D$ represents the *direction* space and $\mathcal{H}^P$ represents the *position* space. $\mathcal{H}^D$ is also commonly referred to as *coin* space (and denoted $\mathcal{H}^C$) as it represents the walker tossing a coin to determine in which direction they will take their next step. $\mathcal{H}^P$ represents all possible positions or states the walker could be in and is analogous to the set of vertices V if the walker is walking on a graph.

Like classical random walks, if standing at position i the walker may cross an edge e_{ij} from position i to position j with some weight w_{ij}. Quantum walks also follow the same methodology as classical random walks wherein the walker first tosses a coin to determine direction and then takes a step at each iteration. The tossing of the coin is represented by a coin operator, written as C, that acts on H^D. For example, consider we start with some arbitrary initial direction $|0\rangle$, where $|0\rangle$ represents the direction *left*, and $|1\rangle$ represents *right*. The coin operator is then applied and yields, for example, $C|0\rangle = |1\rangle$. This means that on the next step, the walker will move to the right.

The shift operator, S, acts on H^P and represents the walker taking a step. Continuing the example from above, given that the outcome of the coin operator is $|1\rangle$, and given an initial position of $|000\rangle$, $S|000\rangle = |001\rangle$ (where position $|001\rangle$ lies directly to the right of $|000\rangle$).

The entire walk can therefore be written as a unitary operation $U = S \cdot C$ where C denotes the coin operator and S denotes the shift operator (note that the coin operator is applied first).

Quantum walks differ in that they can make use of quantum phenomena, namely superposition and interference. Thus, the quantum walker may begin in any superposition of positions, the coin operator may return any superposition of directions, and the walker may consequentially take a step in multiple directions. As a brief example, consider the Hadamard coin, a coin that lands in a superposition of both 'heads' and 'tails' and can be implemented using the Hadamard gate. A walker on a line starting at position $|p\rangle = |0\rangle$ therefore takes a step to end up in the position $\frac{1}{\sqrt{2}}(|-1\rangle + |1\rangle)$, a result of having taken a step both left and right at the same time. Certain coins (such as the Hadamard coin) introduce both positive and negative phases. These opposing phases can destructively interfere with each other, creating unique and sometimes unexpected interference patterns (cf. Fig. 4.1).

In fact, choosing the coin operator is of great importance for a quantum walk as it can significantly affect both the hitting time and interference pattern, leading to different probabilities of reading specific positions at measurement. In a quantum walk search algorithm defined by Ambainis et al. [6, 10], the time complexities of using a 'moving' coin versus 'flip-flop' coin are compared [10]. The moving coin S_m does not change the direction at each step; if the coin landed on direction d in the previous step, it will again land on d. The flip-flop coin S_{ff} 'flip-flops' between two directions: left and right or up and down. Using S_m results in a running time of $\Omega(N)$ for the

walk search algorithm, given N positions arranged in a $\sqrt{N}$ by $\sqrt{N}$ torus. Using S_{ff} on the same torus results in an $O(\sqrt{N}\log N)$ time search algorithm. The S in S_m and S_{ff} represents the shift operator which is defined as $S : |i\rangle \otimes |x\rangle \longrightarrow |\pi(i)\rangle \otimes |\hat{x}\rangle$, where $i \in 1, ..., d$, π is a permutation of directions, and $\hat{x}$ is connected to x by an edge labelled i on x's side [10]. By this definition, the coin operator is equivalent to the permutation π.

$$
\begin{aligned}
S_m : |{\rightarrow}\rangle \otimes |x, y\rangle &= |{\rightarrow}\rangle \otimes |x+1, y\rangle \\
|{\leftarrow}\rangle \otimes |x, y\rangle &= |{\leftarrow}\rangle \otimes |x-1, y\rangle \\
|{\uparrow}\rangle \otimes |x, y\rangle &= |{\uparrow}\rangle \otimes |x, y+1\rangle \\
|{\downarrow}\rangle \otimes |x, y\rangle &= |{\downarrow}\rangle \otimes |x, y-1\rangle
\end{aligned}
\qquad
\begin{aligned}
S_{ff} : |{\rightarrow}\rangle \otimes |x, y\rangle &= |{\leftarrow}\rangle \otimes |x+1, y\rangle \\
|{\leftarrow}\rangle \otimes |x, y\rangle &= |{\rightarrow}\rangle \otimes |x-1, y\rangle \\
|{\uparrow}\rangle \otimes |x, y\rangle &= |{\downarrow}\rangle \otimes |x, y+1\rangle \\
|{\downarrow}\rangle \otimes |x, y\rangle &= |{\uparrow}\rangle \otimes |x, y-1\rangle
\end{aligned}
$$

Through the difference between running times using S_m compared to S_{ff}, it is clear that selection of the coin operator can affect the efficiency of the walk. However, there is also an advantage to using discrete time quantum walks (with a coin register) instead of continuous time walks. Classically, continuous random walks correspond to a relaxation on the requirement for steps to be taken at discrete intervals; the walker instead waits for some time t before taking the next step, with t drawn from a continuous distribution. Ambainis et al. show an example of an application of 'spatial search' where a discrete time quantum walk outperforms a continuous time random walk [10].

2.9.1 Literature Review on Quantum Walks

'Spatial search' is in fact a strong example of the quantum speedup that can potentially be achieved through quantum walks [15, 10, 64]. Spatial search, like Grover's search algorithm, aims to find a marked element from a number of unsorted elements. The difference between spatial search and Grover's search lies in the structure of the search space; whereas Grover's search is assumed to operate on a completely unstructured search space, the elements in spatial search are structured, commonly in a grid or graph. If we consider a search on N elements organized in a $\sqrt{N}$ by $\sqrt{N}$ grid, although Grover's search is known to run in $O(\sqrt{N})$ time, the algorithm does not take into account the fact that the walker may have to move $\sqrt{N}$ elements between oracle queries in the worst case, causing the overall runtime to take $O(\sqrt{N} \cdot \sqrt{N}) = O(N)$ time and thereby losing the quantum speedup [23, 10]. Since the walker in walk algo-

rithms may only move to adjacent vertices, querying two consecutive elements which are $\sqrt{N}$ elements apart never occurs.

Another example applies a quantum walk algorithm to two perfect binary trees both with height n which share identical leaves (cf. Fig. 2.5) [22]. Whereas any classical random walk requires $O(2^n)$ steps to traverse the graph from the vertex labelled 'entrance' to the vertex labelled 'exit', one can traverse the graph in $O(n)$ time using quantum walks [22]. If the 'exit' vertex is the target (i.e., the goal is to measure the 'exit' position with high probability), Childs et al. give a continuous quantum walk algorithm that will measure the target with certain probability in $O(n^2)$ time.

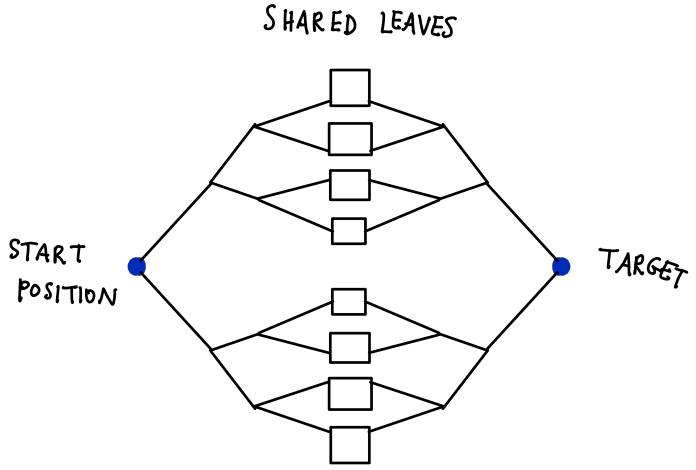


Figure 2.5: Two perfect binary trees each with height n sharing leaves [22].

2.9.2 Quantum Walks and Decoherence

When applying quantum walk algorithms to large systems of qubits, examining experimental data of quantum walks can provide valuable insights into the decoherence process. Alberti et al. [2] analyze an atom moving along a one-dimensional optical lattice and infer the amount of spin and spatial decoherence that takes place at each step of the walk. By comparing such information to existing decoherence models, the primary physical mechanisms by which the decoherence is taking place can also be

inferred.

Kendon et al. [54] prove numerically that permitting a small amount of decoherence can improve the 'desired' effects of coined quantum walks; namely, it can result in faster mixing time for walks on a cycle and faster hitting time for walks on a hypercube.

2.9.3 Quantum Walks on Trees

Although quantum walks are typically carried out on lines, grids, and hypercubes, other structures can be used. Quantum walks on binary trees, for example, open up a different area of applications. One such application uses quantum walks for game theory and reinforcement learning [76, 101]. Since binary trees can be used to represent decision trees (NAND trees) for simple 2-player games, an efficient evaluation of the tree via a quantum walk can identify the best moves a player should take in order to maximize their chance of winning. Mullor et al. [76] show that a quantum walk algorithm for evaluating NAND trees can explore twice as deep as classical algorithms in the same time. Beyond 2-player games, the same binary tree evaluation can be used in reinforcement learning, where an agent must choose an option from a set of possibilities presented to it in order to fulfill some task. In reinforcement learning, the 'best' option will be the one for which the agent receives the biggest reward.

Experimental data using heralded single photons as quantum walkers on central-random glued binary trees (cf. Fig. 2.6) shows the hitting efficiency of quantum walkers from start position to target. When compared to classical random walks, it is shown that the quantum walkers experience improved hitting efficiency as the tree's branching rate increases from 2 to 5 [91]. That is, the more branches there are in the tree, the greater the speedup quantum walks have over their classical counterpart with respect to the time it takes to find the target.

2.9.4 Quantum Walks for Image Segmentation

Just as classical random walks can be used for image segmentation, quantum walks are also able to identify the different components of images. A continuous time quantum walk algorithm is able to identify image segments with approximately 90% accuracy compared to manual segmentation methods [60]. It is proposed that with future work to limit the number of seeds or automatically select seeds, the accuracy can be improved further. However, the algorithm has yet to be translated into a universal

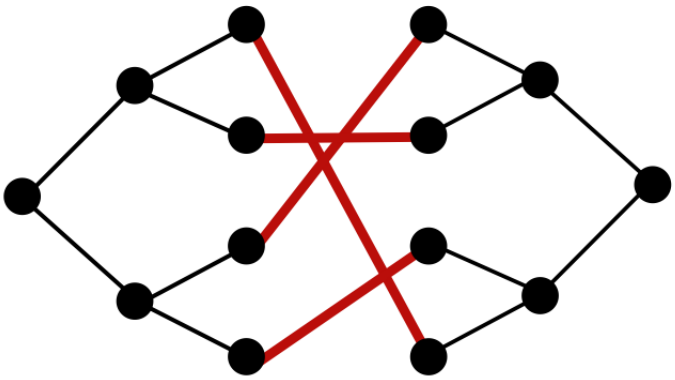


Figure 2.6: An example of central-random glued binary trees. Here, two binary trees each with seven vertices are glued at the leaves via the thick red lines. The leaves are not glued in any order, which is why it is called *randomly glued*.

quantum gate set so that it can be run on real quantum machines. Accurate image segmentation is important in medicine, where components of computed topography (CT) and magnetic resonance imaging (MRI) scans must be identified to diagnose and treat patients [90].

2.9.5 Other Applications

Quantum walk algorithms have also been suggested as having potential with respect to the following topics (among others): element distinctness [7, 63, 47], quantum random access memory (qRAM) [12], webpage ranking [100], image segmentation [60], NP-Hard problems [65, 28, 71, 67], encryption [66], and universal quantum computation [21, 53].

Chapter 3

Quantum Walk Implementations

3.1 General Motivation for a New Quantum Walk Algorithm

Quantum walks have been successfully implemented for many different applications [91, 102, 94, 76, 86]. However, implementations are often at the hardware level, such as [91]; for example, the walker is implemented using an atom, photon, or other quantum particle which moves in an optical circuit [73, 100] or microwave cavity [87]. As a result, these implementations are hardware specific. Although the mathematics and physics outlining the general behaviour of the quantum walk experiment is usually detailed in each article, those whose background lies in computer science (or related field, but not physics) may find it difficult to translate the equations used into a quantum circuit, a description which computer scientists might find more intuitive. Some may wish to replicate the experiment using the quantum computing universal gate set (Pauli gates, controlled gates, and phase gates) so that the quantum walk algorithm becomes hardware agnostic and can be run on any quantum machine without requiring the user to intimately understand how the machine works.

Other implementations ([102, 94, 76, 86]) are shown through circuit diagrams but the circuits are quite long and complex, requiring rotation gates that beginners may find challenging.

3.2 My Motivation

The desire to create a hardware agnostic, generic version of a quantum walk algorithm using the universal gate set lead to the creation of the following algorithms. As described above, it was difficult to find existing quantum circuits for implementing quantum walks using high level quantum gates that were also simple enough for someone new to quantum computing. From the circuits found ([102, 94, 76, 86, 103]), many were already decomposed into rotation gates and were very long [76], were circuits for a specific variant of a quantum walk [86], or otherwise complex and required an involved background in quantum walks and quantum computing research [103]. A simple, high-level algorithm for a generic quantum walk would fill this gap. Furthermore, since it would be high-level, it could be shorter than other circuits and, as a result, easier to understand. Lastly, a generic quantum walk algorithm with only the true basics of a quantum walk (coin and shift operator) would provide a boiler plate algorithm that anyone studying quantum walks could adjust to suit their own specific use case. Thus, the algorithms provided in the following sections attempt to fit exactly this mould of being *simple*, *generic*, and *hardware agnostic*. The Qiskit library was selected due to its widespread popularity. However, one can replicate the algorithms presented here in any programming language that supports quantum circuit building through the universal gate set (e.g., Q#, Cirq, PennyLane).

3.3 Methodology and Challenges

Approaching the creation of the generic quantum walk algorithm from a computer science angle, I decided to break up the walk algorithm into multiple subroutines. Initially, the two primary subroutines appear to be the coin operator and the shift operator; the coin operator determines the direction of the walker, and the shift operator prompts the walker to take a step in the direction given by the coin operator. Thus, two quantum registers are identified, Q_{dir} and Q_{pos}, where Q_{dir} holds the qubits that represent all possible directions and Q_{pos} holds the qubits that represent all possible positions. For example, a quantum walk on a one-dimensional line allows for only two directions the walker can move in: either left or right. In that case, Q_{dir} is a single qubit, since a single qubit is a linear combination of two basis states, and those two basis states represent the two possible directions. Q_{pos} works similarly, only enumerating the possible positions the walker instead of the direction.

3.3.1 Coin Operator

For the coin operator, we initially choose the Hadamard gate. It is easy to implement as a Hadamard gate is simply added to each qubit in the Q_{dir} register (also known as the *Hadamard transform*).

To implement other coins, one needs a different gate or set of gates. Changing the coin operator can be done without affecting the rest of the algorithm, and thus demonstrates *low coupling* — when components of a system operate independently of each other, often desired in software development [50].

3.3.2 Shift Operator

The shift operator requires the walker to take a step in the direction dictated by the coin operator. It is more difficult to implement than the coin operator as it requires knowledge of the structure the walker is moving on. For example, consider the state $|010\rangle$ where the leftmost $|0\rangle$ represents Q_{dir} and the rightmost $|10\rangle$ represents Q_{pos}. Moreover, suppose that $|0\rangle$ has been mapped to the direction *right*. This state indicates that the walker at position $|01\rangle$ should move to the 'right' after the shift operator is applied. But which state is to the right of position $|01\rangle$? What if there is no state to the right of that position? What if there are multiple states to the right? And lastly, how is 'to the right' even defined? For example, if we take a simple connected graph as the structure that the walker walks upon, any two vertices can be moved in space such that one lies to the right of the other (cf. Fig. 3.1).

(a) Vertex A is to the left of vertex B.

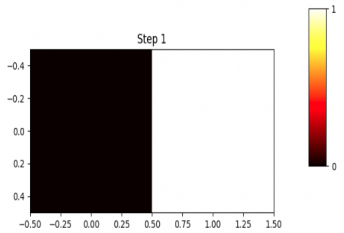
(b) Vertex B is the to left of vertex A.

Figure 3.1: A graph G with two vertices, $V = \{A, B\}$, can be drawn two different ways.

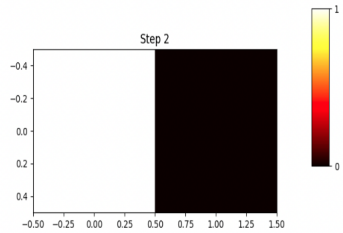
As a result, a walk on a uniform structure that has a clear mapping of directions was quickly identified as being the easiest to implement. Lines and grids have a one-to-one mapping from Q_{dir} basis states to number of available directions. More concretely, a walker on a line can move two directions (left or right) which can be represented using a single qubit. A walker on a grid can move four directions (left,

right, up, or down) which can be represented using two qubits. We do not allow the walker to move diagonally. Precisely because of the efficient and simple mapping between direction space and qubits, lines and grids are popular structures in the quantum walks literature [10, 98, 103, 70], and thus are beneficial for those studying quantum walks to understand.

The next obstacle comes from the finitude of lines and grids; what should happen when the walker reaches the end of a line or border of a grid? Although bounded quantum walks exist [98], it is more popular for quantum walks to take place on repeating structures, such as cycles and toruses [58, 55]. These structures allow the walker to 'walk off' one side of the space and reappear on the opposite side; when visualized in their entirety, these structures 'wrap-around'. Unbounded quantum walks are thus not only popular in literature but also simpler to implement, as one can use a similar circuit to a classical adder or subtractor without the carry. For example, suppose a walker moves along a cycle with only two states, $|0\rangle$ and $|1\rangle$. If the walker starts in position $|1\rangle$, then moving left or right should result in the walker arriving at position $|0\rangle$ (cf. Fig. 3.2), which can be achieved by incrementing or decrementing the binary value 1 without using a carry bit.



(a) The walker starts in position $|1\rangle$ (the white square) with probability $p = 1$.



(b) After taking one step to the right, the walker appears on the left side of the line at position $|0\rangle$ with probability $p = 1$.

Figure 3.2: A walker stands at initial position $|1\rangle$ as shown in (a). After taking a step only to the right, the walker arrives at position $|0\rangle$ as seen in (b). This demonstrates the 'wrap-around' effect programmed into our quantum walk implementations. As a result, a quantum walk on a line is actually a walk on a cycle.

Since incrementing and decrementing are integral to adjusting the position of the walker, the shift operator is further split into subroutines: incrementer and decrementer. When walking on a cycle, one could map the direction $|0\rangle$ to counterclockwise

and $|1\rangle$ to clockwise. If all positions are numbered from 0 to n, then taking a step counterclockwise can be achieved by simply decrementing the value of the current position. Taking a step clockwise means incrementing the value instead.

Incrementing an n-qubit register can be achieved using the following circuit (cf. Fig. 3.3).

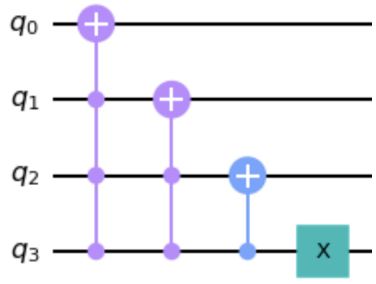


Figure 3.3: A four qubit incrementer circuit. To accommodate n qubits, additional multi-controlled X gates are added to the left. If the $n-1$ least significant digits are all $|1\rangle$, the most significant digit must be flipped. For example, incrementing $|0111\rangle$ results in $|1000\rangle$. Incremeneting $|1111\rangle$ results in $|0000\rangle$ due to torus-like functionality. In both cases, the most significant digit must be flipped. Using recursion, this logic is applied to smaller and smaller subsets of the qubits, until the subset contains only the least significant qubit, which must always be flipped.

Decrementing an n-qubit register can be achieved implementing the incrementer circuit in reverse (cf. Fig. 3.4).

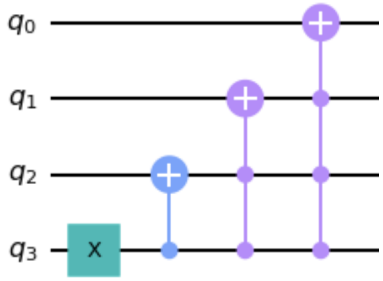


Figure 3.4: A four qubit decrementer circuit. To accomodate n qubits, additional multi-controlled X gates are added to the right.

Note that if the input qubits are in superposition, every term in the superposition is either incremented or decremented at once.

However, when walking on a grid or torus, one must consider precisely which qubits in Q_{pos} to increment and decrement in order to match the direction given by Q_{dir}. Which qubits are incremented or decremented relies on both the numbering

of the positions and the mapping of directions. In a torus, there are four directions in which the walker can move: up, down, right, and left. Assignment of the four directions is arbitrary and works with any one-to-one mapping. Suppose we create the following directional mapping:

$\|00\rangle$	right
$\|01\rangle$	down
$\|10\rangle$	left
$\|11\rangle$	up

and we label positions in incrementing order from $|0\rangle^{\otimes n}$ to $|1\rangle^{\otimes n}$ (where n is the number of qubits in Q_{pos}), starting at the leftmost upper corner and increasing left to right and up to down. Thus, for sixteen positions arranged in a 4x4 torus (cf. Fig. 3.5), positions would be labelled in the following order:

$\|0000\rangle$	$\|0001\rangle$	$\|0010\rangle$	$\|0011\rangle$
$\|0100\rangle$	$\|0101\rangle$	$\|0110\rangle$	$\|0111\rangle$
$\|1000\rangle$	$\|1001\rangle$	$\|1010\rangle$	$\|1011\rangle$
$\|1100\rangle$	$\|1101\rangle$	$\|1110\rangle$	$\|1111\rangle$

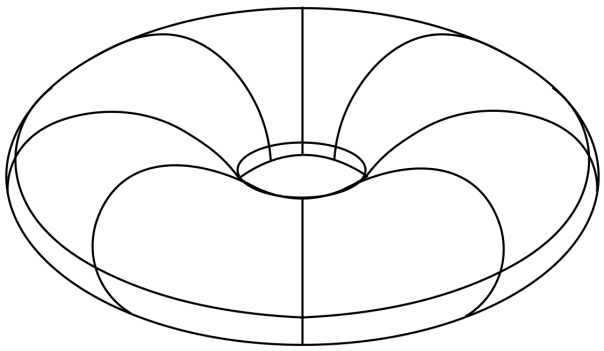


Figure 3.5: A torus with 16 positions.

Since our position space is a torus, taking a step down from position $|1111\rangle$ results in the walker moving to position $|0011\rangle$. Similarly, moving to the right of $|1111\rangle$ results in the walker reaching position $|1100\rangle$.

It can now be seen that the direction in which the walker steps results in the following increments and decrements.

Direction	**Increment/Decrement**	**Qubits**
RIGHT	increment	rightmost two qubits
DOWN	increment	leftmost two qubits
LEFT	decrement	rightmost two qubits
UP	decrement	leftmost two qubits

To try to find a generalization for the number of qubits to be incremented and decremented, we first observe the following specific torus in table form, with a width $w = 4$ and height $h = 8$. We observe that walking horizontally results in an increment or decrement of the rightmost $\sqrt{w}$ qubits, while walking vertically changes the leftmost $\sqrt{h}$ qubits. Note that as a result, both w and h must be powers of two.

$\|00000\rangle$	$\|00001\rangle$	$\|00010\rangle$	$\|00011\rangle$
$\|00100\rangle$	$\|00101\rangle$	$\|00110\rangle$	$\|00111\rangle$
$\|01000\rangle$	$\|01001\rangle$	$\|01010\rangle$	$\|01011\rangle$
$\|01100\rangle$	$\|01101\rangle$	$\|01110\rangle$	$\|01111\rangle$
$\|10000\rangle$	$\|10001\rangle$	$\|10010\rangle$	$\|10011\rangle$
$\|10100\rangle$	$\|10101\rangle$	$\|10110\rangle$	$\|10111\rangle$
$\|11000\rangle$	$\|11001\rangle$	$\|11010\rangle$	$\|11011\rangle$
$\|11100\rangle$	$\|11101\rangle$	$\|11110\rangle$	$\|11111\rangle$

Thus, with all the parts required, the entire shift operator can be constructed. First, X gates are applied to act as conditionals which identify the value of Q_{dir}. For example, if $Q_{dir} = |10\rangle$, an X gate is applied to the rightmost qubit, changing the value from $|10\rangle$ to $|11\rangle$. Note that if the original state of Q_{dir} does not include $|10\rangle$ in its superposition, the resulting superposition after applying the X gate cannot include $|11\rangle$. Thus, applying an X gate to the second qubit will result in $|11\rangle$ if and only if the original state was $|10\rangle$.

This concept is extremely useful and can be used as an analogue to conditional branching in classical programming. Once a $|1\rangle^{\otimes n}$ state is obtained, it can be used as a control for a multi-controlled gate. Furthermore, by adjusting which qubits have X gates applied to them, one can ensure that the controlled gate will only act upon its target if the X gates perfectly align with the $|0\rangle$ qubits in a predetermined original state. If and only if the original state is valid, then a gate is applied to some target.

A possible concern is that by adding X gates to a state, we change it. However, one can easily revert back to the original state by adding a second X gate to each initial X gate. The second X gates are added after the controlled gate is applied, changing the affected qubits back to their initial state (cf. Fig. 3.6).

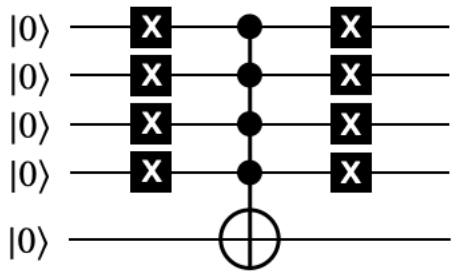


Figure 3.6: A quantum oracle circuit. If the top four qubits are in state $|0\rangle$, then the multi-controlled X gate applies the X gate to the bottommost qubit. At the end of the circuit, the top four qubits are returned to their original state by adding a second X gate to each of them.

The final shift operator changes depending on the number of directions and positions in the quantum walk (cf. Fig. 3.7-3.8).

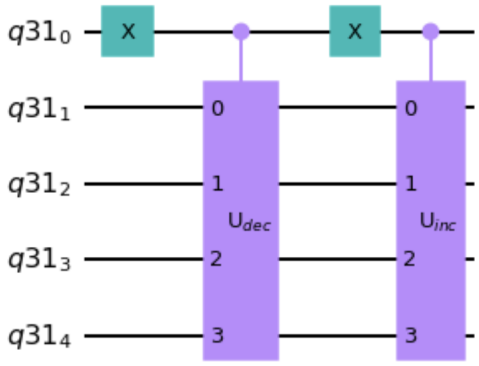


Figure 3.7: Shift operator subcircuit for quantum walks in one dimension. $Q_{dir} = q31_0$ and $Q_{pos} = [q31_1, q31_2, q31_3, q31_4]$. Gates U_{dec} and U_{inc} represent decrementer and incrementer circuits respectively. Note that Q_{pos} will be decremented only if $Q_{dir} = |0\rangle$, and incremented only if $Q_{dir} = |1\rangle$.

3.3.3 A Complete Quantum Walk Algorithm

The coin and shift operators are the only two operators necessary to implement a full quantum walk. The full algorithm is broken up into two main sections: initialization

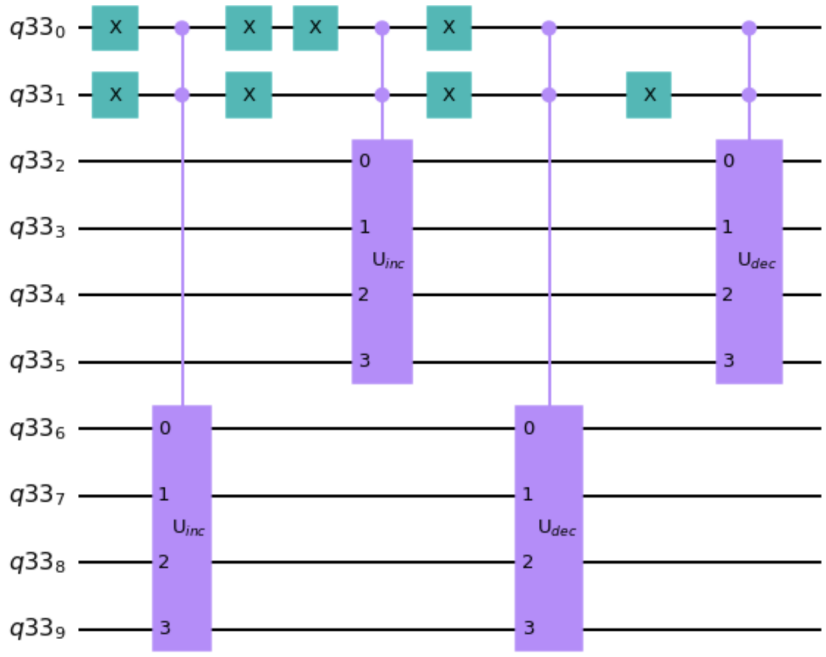


Figure 3.8: Shift operator subcircuit for quantum walks in two dimensions. $Q_{dir} = q31_0$ and $Q_{pos} = [q31_1, q31_2, q31_3, q31_4]$. Gates U_{dec} and U_{inc} represent decrementer and incrementer circuits respectively.

and iterations. During the initialization phase, Q_{dir} and Q_{pos} are initialized.

An initialization of Q_{pos} represents the initial position of the walker. For example, perhaps the walker begins in a corner, or perhaps they begin already in a superposition of positions.

The initialization of Q_{dir} is slightly more difficult to perceive, as the coin operator is run immediately after and will, as a result, immediately change the initial value of Q_{dir} (if the coin is not the identity matrix). However, one common example deals with a quantum walk on a line. Since the walker moves only in one dimension, Q_{dir} is a single qubit. If Q_{dir} is initialized to $|0\rangle$ and the coin is the Hadamard gate, then after applying the coin operator for the first time, Q_{dir} becomes $|+\rangle$. If Q_{dir} were initialized to $|1\rangle$ instead, the result after applying the coin operator for the first time would be $|-\rangle$.

After the initialization phase comes the main part of the walk: the steps. Each step the walker takes is comprised of one application of the coin operator to determine the direction, followed by one application of the shift operator to take a step in said direction. As a result, taking multiple steps can be easily implemented with a for-loop.

Although entire optional, a final measurement of any subset of qubits can be performed at the end of the walk, after all steps are taken.

Walks on Hypercubes

Quantum walks on lines, cycles, lattices, and toruses are the easiest to implement due to the reasons outlined above: namely, that the number of directions in these structures are a power of two. However, walks on hypercubes are also often of interest [84, 57, 99, 49, 29]. Hypercubes pose a slight challenge as they do not guarantee that the number of directions the walker can move in is a power of two.

Take, for instance, a standard cube in three dimensions. Given three axes, the number of directions of movement is six. However, six is not a power of two. Three qubits are required to represent six directions but they offer eight possible values, meaning there are two extra values which must be dealt with. The two extra values can be mapped to any of the already mapped directions (up, down, left, right, forward, or backward), but doing so causes an increased probability to travel in those two extra directions. For example, consider the following mapping.

$\|000\rangle$	up
$\|001\rangle$	down
$\|010\rangle$	left
$\|011\rangle$	right
$\|100\rangle$	forward
$\|101\rangle$	backward
$\|110\rangle$	left
$\|111\rangle$	up

The probability of taking a step down, right, forward, or backward is each $\frac{1}{8}$, while the probability of taking a step up or left is each $\frac{1}{4}$.

The two extra values can also be mapped to self loops such that the walker does not move from their current position. As can be seen, the designation of the two extra values is arbitrary and skews the theoretical probabilities of the walk. The problem can be fixed in two ways: either the two extra values enable the walker to take a step in all (six) base directions with even probability, or one finds a different way to represent directions that does not rely on the power of two domain that qubits offer. We leave this as future work.

3.4 Quantum Walks for Searching

The coin and shift operators outlined in the previous sections can be used to create a generic quantum walk. However, sometimes it is desirable to not merely traverse a space quantumly, but to search and identify a target within that space. This is where turning a generic quantum walk into a search algorithm can be of use.

In order to identify a target element, a third operator must be introduced alongside the coin and shift operators: the *oracle*. The oracle will identify any target position(s), for example, by setting an ancilla qubit to $|0\rangle$ if the input is not a target, and $|1\rangle$ if the input is a target (cf. Fig. 3.9). Using the output of an oracle and determining what kind of output the oracle should have is based on preference and use case. In this thesis, I will use ancilla qubits to implement oracles as they provide a simple and generic solution.

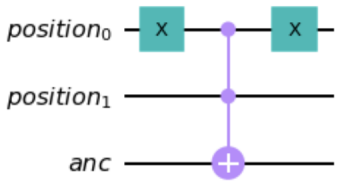


Figure 3.9: An example of a boolean oracle. Let the target position be $|01\rangle$. We assume the ancilla qubit starts in state $|0\rangle$. Then the ancilla only changes to $|1\rangle$ if and only if the position qubits are in state $|01\rangle$.

Therefore, in addition to the Q_{dir} and Q_{pos} registers, another qubit is added and labelled q_{anc} (for ancilla qubit). The oracle takes as input the current position of the walker and sets q_{anc} as output. The oracle itself is positioned within the for-loop which represents the walker iteratively taking steps. The oracle should be the first operation applied at each iteration. Refer to Fig. 3.10 for a short comic on how the oracle works during a quantum walk.

However, a challenge arises when setting q_{anc} iteratively. Suppose the walker is lucky and begins at a target position. The oracle is applied and will set q_{anc} to $|1\rangle$. The coin operator is then applied, followed by the shift operator. Assuming the walker moves to a new position, q_{anc} will incorrectly remain set to $|1\rangle$ in the new position. Instead, q_{anc} should be $|1\rangle$ only when the walker is at a target position, and should the walker leave a target position, q_{anc} should return to $|0\rangle$.

Instinctively, it is tempting to think a simple X gate could solve this issue. After the walker takes a step, an X gate could be applied to return q_{anc} to $|0\rangle$. However,

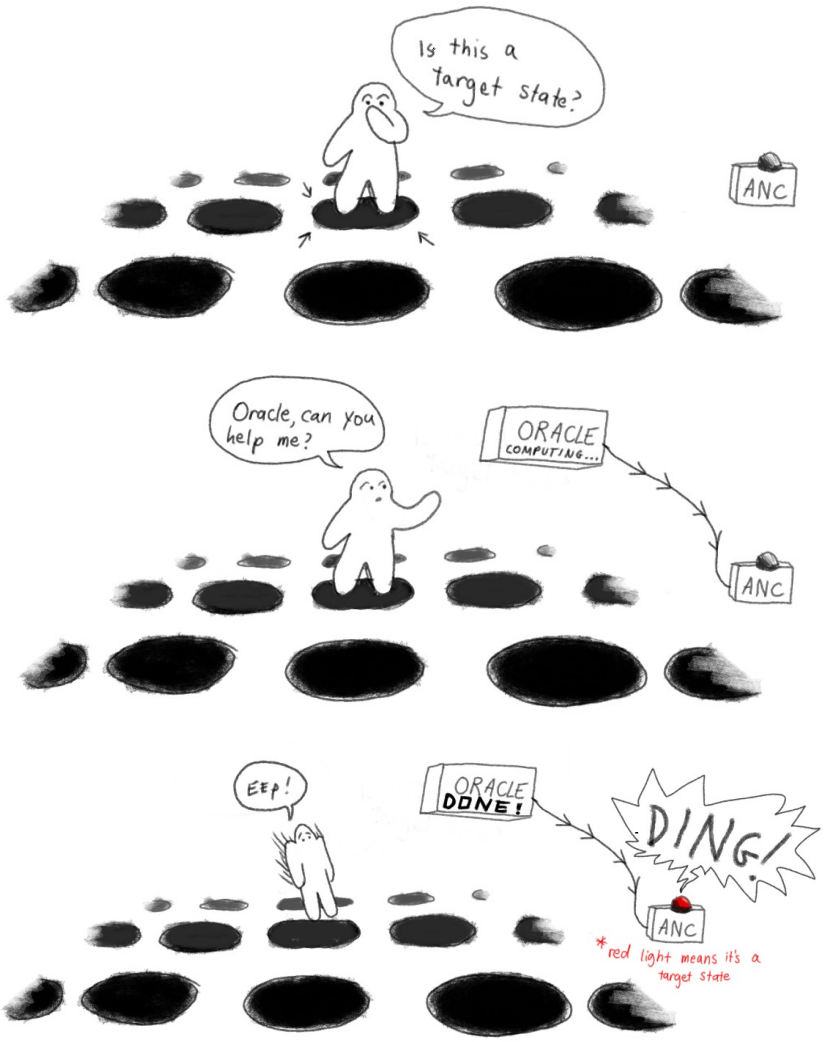


Figure 3.10: The dark circles on the ground represent positions that the walker can step to. Every time the walker takes a step, they ask the oracle if their new position is a target state. The oracle replies by setting the ancilla to either $|0\rangle$ or $|1\rangle$, where $|0\rangle$ means it is not a target state, and $|1\rangle$ means it is a target state. In this comic, the ancilla is represented by a light which remains off if it is not a target state, and which lights up with red light if it is a target state.

suppose the walker begins at a non-target position and q_{anc} is initially set to $|0\rangle$. After applying the oracle, q_{anc} will remain as $|0\rangle$ since the walker is in a non-target position. Applying an X gate after the walker takes a step will result in q_{anc} becoming $|1\rangle$, which is likely incorrect if the walker's new position is also a non-target state, since $|1\rangle$ indicates a target position.

Following this path, one might try to correct the behaviour by changing the X gate

into a controlled X gate; then only if $q_{anc} = |1\rangle$ will the X gate be applied. However, that would mean q_{anc} would be both the control and target, and when using controlled gates recall that the target qubit(s) cannot include any control qubit(s).

For our generic quantum walk, a simple solution of swapping out 'used' ancilla qubits was put into place. After each step of the walk, the current ancilla qubit is swapped with a qubit set to $|0\rangle$. Thus, for a quantum walk with t steps, a total of t ancilla qubits must be prepared.

3.4.1 Ensuring Measurement of the Target

With the ancilla qubits in place, the next goal is to measure the target state with reasonably high probability. While it is possible for quantum walks to search N elements in $O(\sqrt{N})$ time [10, 108] (compared to the $O(N)$ time required classically), the algorithms that achieve that time complexity may be difficult to understand for beginners. Quantum walk search algorithms often make use of quantum subroutines (Grover's search and quantum phase estimation), and the walks are typically edge-centric instead of vertex-centric as one may intuitively imagine; random walks and other quantum walks feature the walker resting at vertices instead of on edges. It is also difficult to visualize quantum walk search algorithms due to the nature of amplitude amplification, the use of edges, and the size of the graph required to simulate multiple iterations of the walk.

Sticky Walk

Inspired by the concept of 'hit-boxes' as described in [9] and the self-loops added to graphs in [104] (cf. Fig. 3.11), I introduce a new quantum search walk algorithm called 'sticky walk'. Here, the walker 'sticks' to the discovered target(s) and does not leave upon further iterations of the walk. Sticky walk requires the use of an auxiliary qubit, which marks target states at the beginning of each iteration and which is reset at the end of each iteration by swapping to a 'fresh' qubit as described in Appendix 3.4. Algorithm 1 outlines sticky walk in pseudocode.

(a) Random walk with probability of staying at a target $p = 0$.

(b) Random walk with probability of staying at a target increased to $p = 0.5$.

Figure 3.11: Consider a random walk denoted by a series of white and red boxes. White boxes represent the walker in a non-target state; red boxes represent the walker in a target state. In (a), the walker begins in a non-target state and after three steps, enters a target state. The walker then leaves the target state and never enters another target state for the remainder of the walk. In (b), a self-loop is added to all target states, which increases the probability of staying in a target state from $p = 0$ to $p = 0.5$. As a result, we can expect the walker to stay in a target state on average for twice as long. Increasing the number of self-loops in target states will further increase the probability of the walker remaining at target states.

Algorithm 1 Sticky walk algorithm pseudocode

1: **procedure** StickyWalk($G = (V, E)$)
2: $q_{pos} \leftarrow$ [initial walker position]
3: $q_{dir} \leftarrow$ [initial coin superposition]
4: $q_{anc} \leftarrow |0\rangle$
5: **for** `numIterations` **do**
6: $q_{anc} \leftarrow |0\rangle$
7: **markTargets**(q_{anc}, q_{pos})
8: **applyCoinOperator**(q_{dir})
9: $X(q_{anc})$
10: **applyControlledShiftOperator** ($q_{anc}, q_{pos}, q_{dir}$)
11: **end for**
12: $m \leftarrow$ **measure**(q_{pos})
13: **return** m
14: **end procedure**

First, all targets are marked using an oracle (line 7) such that q_{anc} is set to $|1\rangle$ on target positions and to $|0\rangle$ otherwise. Next, the coin operator is applied (line 8) to put q_{dir} into any desired superposition. A controlled shift operator is then applied (line 10), where only non-target states are allowed to shift to new positions. Target states are not shifted, which means that once a target state is discovered by the walker, the walker is not able to leave that state. After sufficient number of iterations, the

position qubits are measured (line 12) and the result returned (line 13).

The efficacy of sticky walk relies heavily on the coin operator. Walks that cover a large portion of the search space perform best as the walker is likely to reach (and stick to) the target regardless of initial position. The running time of sticky walk relies on both the coin chosen and the desired probability of measuring a target state.

3.5 Open Source Code

Code for two-dimensional and three-dimensional quantum walks can be found at the URL: `https://github.com/addie43110/qwalk-visualizations`. The code is written in a Jupyter notebook so users can walk through each step of the various quantum walks and examine each subroutine individually. For those familiar with Jupyter notebook and Python, running the code is very easy. Additionally, each code block in the notebook can be modified and run multiple times, allowing the user to test out their assumptions and check the output to see if the two align.

However, for those with little to no programming background or who are unfamiliar with Jupyter notebook or Python, the notebook can be difficult to set up as it requires multiple imports and the setting up of a programming environment. Furthermore, looking at the code requires knowledge of basic programming in order to understand things such as variable declaration, types, and conditionals, which are necessary concepts to understand the quantum walk algorithms as a whole. Thus, we created the website application QWalkVis (cf. Chapter 4) so that one can visualize and run quantum walks with zero programming knowledge required.

Chapter 4

Quantum Walk Visualizations

As a student for several years, I have almost always found visualizations immensely helpful, particularly in lectures full of dense mathematical notation. Even a quick diagram or brief sketch often helped me understand, for example, how an algorithm might traverse a higher-dimensional domain or what was happening to a curve when multiplied by a constant. As someone who has taught consistently for many years as well, I have noticed students perking up when I am able to equate a verbal or mathematical definition to a visualization. For example, I often use the trick of the Venn diagram visualization to describe the basics of set theory, namely union, intersection, and complement. While I can only speak to my own experiences, I believe that good visualizations (alongside many other solid teaching practices) can be of great help and, at the very least, cause no harm when used alongside already established educational material such as lecture slides and textbooks (pending their correctness and complete separation from any harmful biases).

4.1 Use of Visualizations in Education

Visualizations and media such as images, videos, animations, and interactive games have become increasingly accessible as the technology assisting research (e.g., microscopes or simulation software) has continued to advance. Biology is one area that has benefited from using visualizations as a core part of its education culture as many foundational biological processes occur at the sub-microscopic level [48]. Earth scientists can use animations to display what might take thousands of years to play out in mere seconds. Physics simulations abate not only curiosity for how a brick might

slide on a friction-less surface, but provide educators with a visual representation of the underlying mathematics explained in class.

Visualizations are regarded by many as helpful in education which can be seen in the abundance of illustrations in textbooks, in some cases comprising more than 50% of the material within [48]. Some even predicted that the onset of radio and television would solve many educational problems [27, 42], and the 'Picture Superiority Effect' is a documented phenomenon that claims there exists a recognition advantage for imagery over word stimuli [14]. However, visualizations alone will not result in efficient, holistic understanding that is often desired. Poorly designed illustrations, illustrations with too much information, and visualizations designed with no specific goal can prevent the learner from being able to hone in on the specific concepts and explanations required to learn an idea fully [82]. Well-constructed visualizations with meaningful associated text have the potential to help learners with recall [14], and can therefore act as an aid in the learning process alongside other forms of both passive and active learning.

It is important to discuss accessibility of visualizations as well. It has been suggested through personal account that people with autism may 'think in pictures' [37] – visual mental constructs replacing the verbal processes that neurotypical adults rely on for typically verbal tasks such as serial recall [61]. Including a visual representation of an idea in addition to a verbal or written explanation could potentially assist those who prefer to use visual models as they may find it easier to encode in memory. Visualizations and animations may also cost less to produce than an in-person demonstration, for example the website Quantum Flytrap that allows students free access to a virtual photonics laboratory [34] and run their own experiments, which may otherwise cost hundreds to thousands of dollars to access. Images and animations may also optionally include noise, closed captioning, alternative text, and heightened contrast allowing students to adjust such options to fit their needs.

4.2 Existing Quantum Walk Visualizations

Understanding both the mathematical notation of quantum walks and connecting the mathematics to the visualizations are both important when it comes to fully comprehending quantum walks. In addition, the visual representations may help beginners who do not (yet) have sufficient background in mathematics, physics, or computer science. Although there are many visualizations of probability distributions

created by quantum walks [8, 98, 83, 52, 6] (cf. Fig. 4.1), attempts to visualize the walker moving 'quantumly' through space are fairly scarce. One method of visualizing classical random walks is using the *quincunx* (also known as Galton's quincunx or Galton's board), where marbles are funnelled and then dropped through a vertical series of equidistant nails, resulting in a Gaussian distribution-type pile at the bottom of the device [51, 19, 83] (cf. Fig. 4.2). The marbles represent different walkers and the nails outline different paths the walker could take. Furthermore, when a marble reaches a nail, it is random as to whether the marble will fall to the left or to the right, demonstrating the randomness of the path the walker takes.

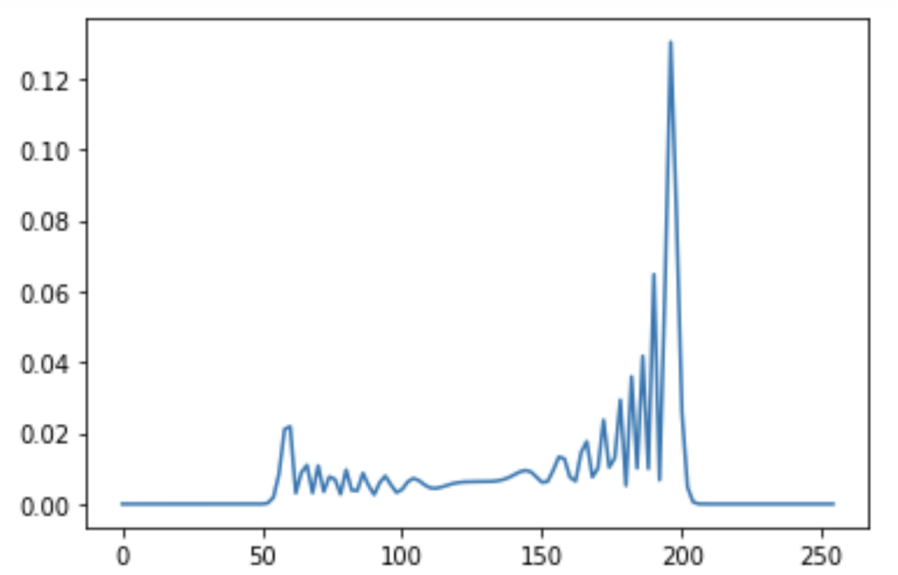


Figure 4.1: The probability distribution of a quantum walk on a line using a Hadamard coin after taking 100 steps. The x-axis enumerates the positions on the line with left-most position 0 and right-most position 250. The y-axis shows the probability of the walker being in a given position. The walker starts at initial position 125 and takes steps both left and right 100 times. The highest probability occurs approximately at position 200, where the walker has a probability of $> 12\%$ of being at that position after taking 100 steps.

A digitalized version of the quincunx could offer an interactive playground for learners to adjust parameters of the walk and witness the effects. For example, the learner could slide the funnel left and right to change the initial starting position of the walker. Rows of nails could be added or removed to simulate the number of steps the walker takes. Start and stop buttons could allow the learner to pause the simulation at any point and inspect the current state. A three dimensional version of

the quincunx could even be constructed such that when the marbles reach the bottom of the device a two dimensional distribution is formed. Each marble could make a copy of itself when it reaches a nail. One copy of the marble would fall to the left, the other to the right, representing the concept of superposition.

A different version of the quantum quincunx has already been proposed and uses a single atom traversing through a microwave cavity to demonstrate a quantum walk on a circle [87]. However, it does not demonstrate walks on higher dimensions and requires access to specialized equipment.

Although the quincunx could be modified and digitized to visualize quantum walks, it may still cause confusion for the learner trying to map concepts between the device and random walks. For example, a learner may not understand why the marbles are being funnelled in the first place; they are funnelled because the walker starts at a single location before deciding to step right or left. Learners may also not understand what the use of multiple marbles suggests; multiple marbles can be used to represent multiple different walkers. Even when the marbles start at the same initial location, they make bounce on different nails, demonstrating the randomness of the walk. The full lines of 12 and 13 nails in the first few rows may suggest that it is somehow possible for the marbles to reach the outer left and right sides of the device in only a few steps which is in reality impossible if the marbles are dropped in the middle of the first row. Walks on two- and three-dimensional spaces are also significantly more difficult to visualize using the quincunx.

As mentioned above, the quincunx itself is usually written as a visualization for classical random walks. There is little in literature and educational material for possible visualizations for quantum walks. Popular online websites for learning quantum computing such as the Qiskit Textbook [25] and Xanadu Quantum Codebook [3] do not offer visualizations for quantum walks aside from circuit diagrams and distribution results.

An excellent online website [81] allows users to view an animation of the quincunx working in real time. The probability that each ball bounces to the left or right of the current peg can be adjusted using a slider. At the bottom, possible final positions (separated by vertical bars in Fig. 4.2) keep tally of the number of balls that have landed in each position.

Such an animation tool is an excellent starting place for visualizing classical random walks. However, it does not discuss quantum walks or quantum theory. One could create their own version of a ‘quantum quincunx’, where each ball can exist

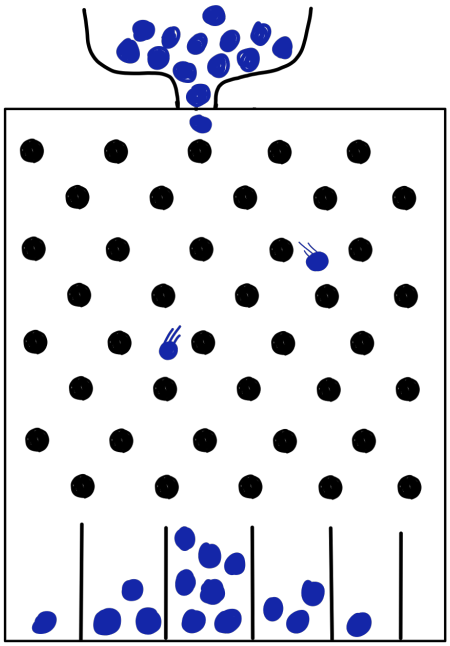


Figure 4.2: Galton's quincunx. The path of each marble is equivalent to the steps taken in a classical random walk. The resulting pile of marbles at the bottom follows the Gaussian distribution.

in a superposition of positions (travel both left and right of the current peg at each bounce). One could also use a colourbar similar to the one in QWalkVis to colour the balls according to their respective probability in the superposition.

Resources specifically for quantum walk visualization include Wolfram Alpha's Quantum Random Walk [44], Cirq's Jupyter Notebook for generating quantum walks [24], and Hiperwalk [38]. All three resources visualize the probability distribution created by running a quantum walk on a certain number of qubits and with a certain number of steps. This differs from QWalkVis, which visualizes the positions themselves as a grid or cube, where each square or cube is coloured according to the probability that the walker is in that position at a given step. Furthermore, such resources only create one visualization which shows the probability distribution after the entire quantum walk is run; QWalkVis creates multiple graphs which show the walker moving through space, which states the walker is in at each step, and the interference patterns created by the walker. Lastly, these resources require the user to have pre-existing programming knowledge to create the visualizations; QWalkVis aims to eliminate that requirement.

4.3 Quantum Walk Visualization Web Application

The quantum walk visualization web application is a website application written in Javascript and Python. The React [72] Javascript framework was used to write the front end components, while the back end uses Python's Flask [79]. The probabilites for each walk are calculated using Qiskit's statevector simulator. The link to the front end Github repository is here. The link to the back end Github repository is here. The application was developed with help from fellow students Samantha Norrie, Austin Hawkins-Seagram, and José Ossorio.

4.3.1 Description

The quantum walk visualization application is a single page web application. It consists of three main parts: a landing page (with contact information and resources), an instructions page, and the visualization component. The instructions page offers a brief explanation of the quantum walks along with links to other resources for learning classical and quantum walks. It also provides brief instructions for how to use the visualization component.

The visualization component itself is comprised of two sub-components: the graph display which plots the desired quantum walk and the options panel (cf. Fig. 4.3) where the user can choose from the following options:

- dimension: the user can select using radio buttons whether the walk should take place in one, two, or three dimensions. That is, on a line, grid, or cube.
- number of states: the user can enter through text input the total number of states the walk should be carried out on, essentially defining the search space.
- number of steps: the user can enter through text input the number of steps the walker should take

It should be noted that all inputs are sanitized and validated and appropriate error messages are shown in real time to the user if an invalid input is entered (cf. Fig. 4.4). Upon submitting the options through a form, a request is sent to the back end with the desired parameters. A plot of the positions the walker is in and associated probabilities is made for each step of the walk including a step zero, which is before the walker has taken a single step.

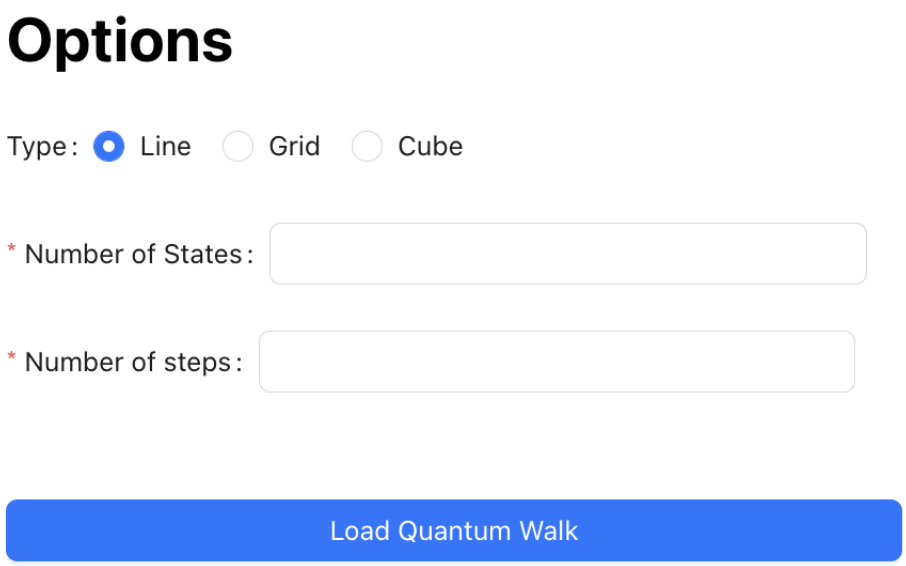


Figure 4.3: The options component where users can select the dimension of the search space, the number of states, and the number of steps the walker takes.

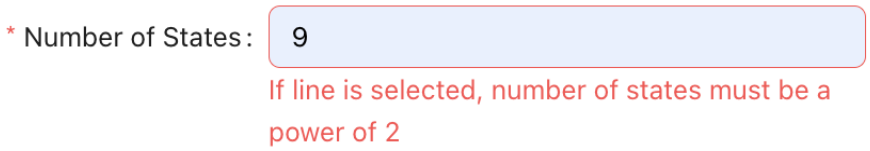


Figure 4.4: Real time validation of user input.

The plots for different steps of the walk are accessed through a slider, allowing the user to jump to and compare different stages of the walk. Probabilities are shown on a normalized colourbar to the right of each plot.

4.3.2 Methodology

QWalkVis was created with the following goals in mind: to showcase quantum walks through novel visualizations, to be interactive, and to provide resources for further study of quantum walks. As such, QWalkVis was built to be quite simple, focusing on the base visualizer with the option to add additional features at a later date.

Keeping with simplicity, a single page web application was determined to be the easiest to build and also most responsive. The React framework was chosen as it is adept at creating responsive single page applications. For the backend, existing

quantum walk algorithms were already implemented in Qiskit (Python), so it made sense to choose Python-native Flask for ease of portability.

4.3.3 Limitations and Future Work

Currently, QWalkVis does not exhibit measurement results, since it uses a statevector simulator to determine each position's probabilty. However, it would be possible to run each quantum walk on a simulator or real quantum backend and plot the results from a single or multiple shots.

As the probabilities for each step of the walk are created using simulators, walks with large values for the number of steps and states can take multiple minutes to load. Furthermore, there is a limit to the number of qubits supported by the simulator which scales with the amount of memory allocated to the process. On a personal computer with eight gigabytes of random access memory (RAM) and eight cores running on default settings, the limit is approximately 42 qubits. Since the visualizations show the probability that the walker is at a given position, a real quantum computer cannot be used to show intermediate state probabilities, although it would be possible to show the end measurement result. Lastly, when walking in a cube, the walker can step in a total of six directions: left and right, up and down, and backward and forward. Since six is not a power of two however, when representing the number of possible directions in binary there remain two extra 'directions'. In the application, those two extra directions have been mapped to left and right again, resulting in higher probabilities for the walker to step left or right. A note was made in the instructions panel to inform the user of this phenomenon.

All walks regardless of the dimension operate on a torus. That is, if the walker walks off of one side of the graph, they will reappear on the other side. The torus was chosen due to its ease of implementation using Qiskit. An alternative version of the walk is to refuse the walker to walk off of any side of the graph. For example, a walker at the leftmost position of a non-infinite line will remain in the same position when told to take a step to the left. Such an implementation can be added with multiple Toffoli gates and, depending on the implementation, additional qubits.

It is important to note that the application requires the length of each side to be a power of two. More explicitly, a walk on a line requires the number of states to be a power of two. A walk on a grid requires the square root of the number of states to be a power of two, and a walk on a cube requires the cube root of the number of

states to be a power of two. Again, this was done for ease of implementation. The minimum input validation required for implementation using qubits is the following: the number of states must be a power of two and

- if a walk on a grid, the number of states must be a perfect square
- if a walk on a cube, the number of states must be a perfect cube

Lastly, lines, grids, and cubes were chosen as they are highly symmetrical and regular spaces. The walks simulated in the application could also be carried out on any k-regular graph with the requirement that k is a power of two and the number of vertices is also a power of two. For non-regular graphs, quantum walks become much more difficult to implement as the coin operator requires knowledge of the number of neighbours $|u|$ each vertex has and must create a superposition of exactly $|u|$ states, which is exceedingly difficult to do when $|u|$ is not a power of two. (If $|u|$ is a power of two, the Hadamard transform can achieve this). Once such a transform is possible to implement in Qiskit, the shift implementation can easily be altered to allow quantum walks on non-regular graphs.

Chapter 5

Quantum Walks and QWalkVis for Education

Although QWalkVis can be used in many different ways and with different goals, a few use cases are outlined in this chapter regarding its envisioned usage.

QWalkVis is primarily intended to be used as a supplementary tool alongside more detailed instruction for teaching students about foundational quantum theory and quantum walks. Although a brief description of quantum walks and resources for quantum walks and quantum theory are given on the website, QWalkVis explains neither the linear algebra nor Dirac notation behind what the user witnesses in the produced plots. It is therefore recommended that QWalkVis be explored alongside lecture notes or textbooks which can explain to the user the underlying quantum theory of the quantum walks.

However, if the intent is merely to generate interest about quantum computing and not understand the mathematics behind it, QWalkVis can be used as a standalone application. Its interactive design and the visualizations it generates can be used to attract attention. Furthermore, since it is a website application, it requires no knowledge of programming nor quantum theory in order to adjust the parameters and explore changes in the resulting plots. As a result, users from any background may find it intriguing to explore QWalkVis and make their own conjectures as to how the plots are produced and what they mean.

Lastly, an expanded version of QWalkVis with added parameters (i.e., number of walkers, starting location, and coin operator) can serve as a tool not only for education, but also research. As mentioned in Chapter 4, it can be difficult to find

visualizations for quantum walks. QWalkVis allows researchers to create their own plots for custom walks on lines, grids, or cubes. Those plots can then be used in presentations or merely as a means for the researcher themselves to visualize and understand the patterns that arise from large or complex quantum walks.

5.1 Education

The main intended use case of QWalkVis is to be used as a supplementary tool alongside more formal and detailed instruction. One such example is usage in a university course setting. Suppose for the next few examples that a lecturer is teaching a course titled 'Introduction to Quantum Computing'.

5.1.1 Live Demonstration

The strength of these visualizations lies in the fact that linear algebra does not need to be discussed in order for students to become acquainted with basic quantum theory. Although linear algebra should be discussed at some point in order to understand the material deeply, initially removing that mathematical barrier may help students from all backgrounds to create their own high-level understanding before delving into details.

During the first or second lecture of the semester, the lecturer can use QWalkVis as a live demonstration in order to generate interest about quantum computing, while also describing at a high level the concepts of superposition, interference, and measurement. In this use case, the lecturer will first give a brief introduction to classical random walks. Consider the following quotes as examples for what the lecturer could say.

> "Consider a person that merely flips a coin every time they come to a fork in the road. If the coin lands heads, the walker goes right. If the coin lands tails, the walker goes left. This is the idea behind classical random walks."

Explaining Superposition

The lecturer can then go on to introduce quantum walks and the notion of superposition.

> "Quantum walks follow the same idea as classical random walks, except quantum walks can use what is called superposition. Superposition means that the coin can land both heads and tails up at the same time, and the walker can travel both left and right. That is, the coin and walker can exist in multiple states at the same time."

While such an explanation is certainly simplified, it quickly gets across the main point of superposition, which is that a quantum state exists in a superposition of discrete states. More detailed explanations of amplitudes and probabilities will follow in subsequent lectures. Furthermore, using simplified language on the first or second lecture will help retain students who may be intimidated by jargon or mathematics. Lastly, the statement that a coin could land both heads and tails up at the same time is quite preposterous to anyone new to quantum, and therefore should generate excitement for the topic.

The lecturer can then use QWalkVis to provide a visualization of superposition by plotting a quantum walk on a line with n positions and at least one step. Before the walker takes a step, the walker is located at a single position on the line (with 100% probability, although that may or may not be discussed on the first lecture) (cf. Fig. 5.1).

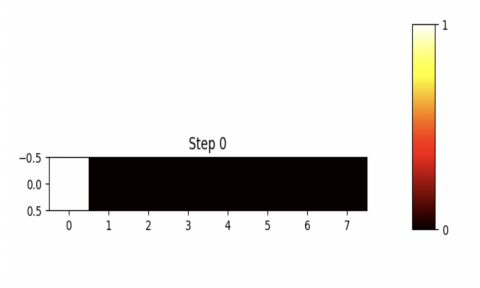


Figure 5.1: A quantum walk on a line with 8 positions. Before taking any steps, the walker is in the initial position with probability 1.

The coin is then tossed and will land both heads and tails up at the same time. This outcome prompts the walker to take a step both left and right at the same time. Since the walk occurs on a cycle, the walker taking a step to the left appears on the far rightmost position. The resulting plot on QWalkVis shows the walker now in two positions, thereby providing a visualization for the concept of superposition (cf. Fig. 5.2).

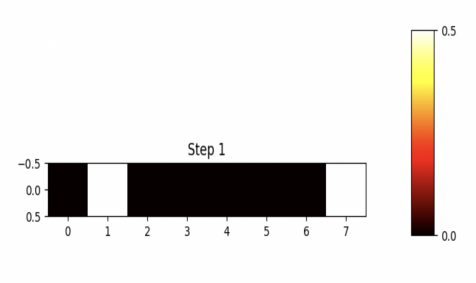


Figure 5.2: A quantum walk on a line with 8 positions. After taking a step, the walker ends up in a superposition of two positions, each with probability 0.5.

Explaining Interference

The lecturer can then produce plots of any quantum walk with at least two steps to demonstrate the concept of constructive interference. However, a walk on a line is recommended, as it is the simplest with fewest possible directions.

For example, consider a quantum walk on a line with 16 states and two steps. After the first step, the walker will be in a superposition of two states (cf. Fig. 5.3).

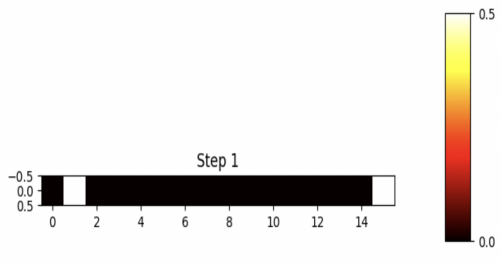


Figure 5.3: A quantum walk on a line with 16 positions. After taking a step, the walker ends up in a superposition of two positions, each with probability 0.5.

Since the walker is now in two states with equal probability, a student may guess that the walker will continue to be in multiple states all with equal probability. The student may then suggest that after taking yet another step, the quantum walker will now be a superposition of three distinct positions, each with equal probability $p = 0.33$. However, what is witnessed is quite different as not all positions in the superposition have the same probability (cf. Fig. 5.4).

The lecturer can explain the fallacy by referring to the states in step 1, where the walker is in a superposition of two positions. On the next step, the walker will take a step in both directions: left, and right. This means each state in Fig. 5.3 is succeeded by its two neighbouring states in the subsequent plot (in the same way

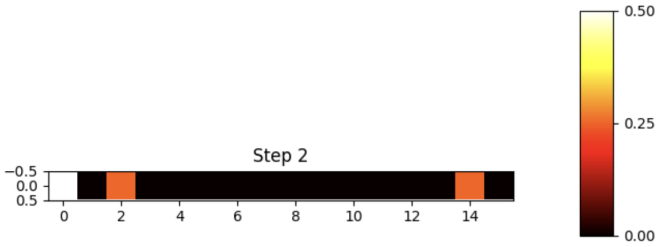


Figure 5.4: A quantum walk on a line with 16 positions. After taking two steps, the walker ends up in a superposition of three positions, two of which have probability $p_1 = 0.25$ while the leftmost position has a probability of $p_2 = 0.5$.

that the initial singular position of the walker 'split up' into the two states seen in Fig. 5.3). Since each position in step 1 has a probability of 0.5, taking a step in two directions will result in the neighbouring two positions each having a probability of $\frac{0.5}{2} = 0.25$. However, both positions have position 0 as a neighbour, and thus in step 2, position 0 will have a total sum probability of $0.25 + 0.25 = 0.5$. This sum of probabilities is exactly constructive interference.

Although this explanation requires an understanding of probabilities, a simpler explanation can be given. Without discussing exact probabilities, one can show that the two positions in step 1 have position 0 in common as a neighbour. Thus, when the walker takes a step in all directions, one 'step' will land on position 2, one on position 14, and two on position 0. The can be drawn using arrows as in Fig. 5.5.

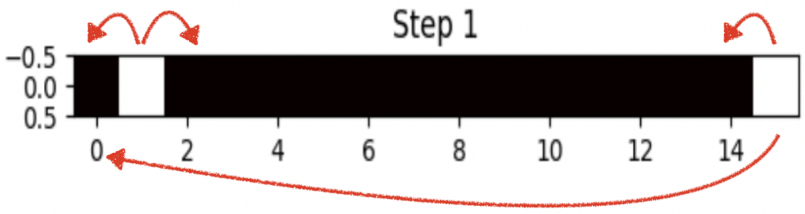


Figure 5.5: The red arrows show the successor states after taking another step. Notice that there are two arrows pointing to position 0. We can use this intuition to explain that position 0's resulting probability will be twice as large as position 2 and position 14, which only have one arrow pointing to them each.

Explaining Measurement

Measurement is the means of extracting information from a quantum circuit. It is often thought of as 'dequantumization' as it takes the final state of a quantum circuit and returns only classical information. In terms of a quantum walk, the walker is likely to be in a superposition of multiple positions at the time of measurement. Thus, measurement will return one position at random drawn from the probability distribution given by the amplitudes in the superposition. This is also referred to as 'collapsing the superposition'.

However, such an explanation is loaded with technical terms and does not give a visualization or concrete example. Using QWalkVis, a lecturer can bring up a plot of a quantum walker in superposition (cf. Fig. 5.6). Since measurement outcomes are entirely dependent on probability distribution, it is highly recommended that discussions of measurement take place only after probabilities have been introduced.

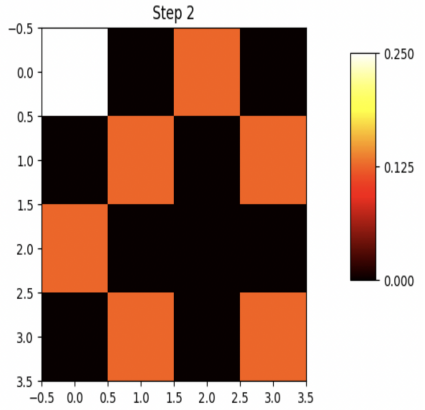


Figure 5.6: A quantum walk on a torus with 16 positions. The plot shows the probability distribution after the walker has taken two steps. The initial position of the walker was the top-left square (the white square).

Using the plot, the lecturer can explain that after measurement, the walker can exist in only one discrete position; that is, the walker can no longer exist in superposition. Further simplified for younger students, the walker being in superposition is akin to not truly knowing where the walker is. However, we have some educated guesses of where the walker could be (the orange and white positions), and how likely the walker is to be in any of those positions. Measurement is then akin to formally finding out where the walker actually is.

The black positions have a probability of zero, meaning there is no chance that

the walker could exist on a black positions. Thus, it is not possible to measure a black position. The white position has the highest probability out of all the positions on the grid, and thus is the most likely position that the walker is in. However, it is important to remember that one cannot know where the walker is until they look; put another way, measurement outcomes are truly random.

5.1.2 Student-led Exploration

Alternative to a live demonstration, an educator may choose to let students explore QWalkVis with no guidance at first, and later check the understanding of the students after a certain allotted time period. Such an approach benefits students who learn best by 'doing'; students who are eager to solve problems by themselves.

The exploration session is suggested to take place further into the semester, or after basic Dirac notation has already been introduced. This allows students to already have the toolkit required for understanding quantum walks (Hadamard gate, controlled gates, etc.) but provides the challenge of working out the different components needed for a quantum walk themselves. For example, students may quickly identify that the walker travels via superposition in all directions. A student who knows of the Hadamard gate may suspect that it is likely used in the underlying algorithm, yet simply applying the Hadamard gate to all qubits will not yield the correct result that the quantum walk shows. This creates a nice problem that is difficult but within reach of the student's capabilities. The exploration session can also take place at the beginning of the semester, but it may be more difficult for novice students (particularly those in high school or younger) to grasp what is occurring. Thus, students may lose interest more quickly.

An exploration session is also recommended at events where time with users is limited. For example, QWalkVis could be set up on computer(s) at a conference, and attendees can come by and explore the application on their own. This offers a talking point for the host(s) to engage in conversations with attendees, and to provide explanations if the attendees are interested.

5.1.3 Additional Educational Materials

Please refer to Appendix A and Appendix B for lecture slides on introducing the foundations of quantum computing and quantum walks.

5.2 Research

As further elaborated in Chapter 7, QWalkVis is still in development, and there are many additions that are being considered. Although the current version of QWalkVis can be used in research to generate visualizations of quantum walks on lines, grids, or cubes, future versions of QWalkVis will aid researchers even more. The addition of allowing parameters such as initial position, number of walkers, and coin operator to be changed will allow for highly customized quantum walks visualizations to be produced.

Since the walker can only exist in a superposition of all odd or all even states, the maximum number of distinct positions in the superposition is $\frac{N}{2}$, where N is the total number of positions. For example, a quantum walk on a line with eight total positions means the walker could be in a maximum of four positions at any given time. As such, for walks where N is large, it can be exceedingly difficult to calculate the amplitudes for each state in the superposition by hand. QWalkVis can alleviate researchers from having to do manual calculations as it lists the probabilities of each position in the colourbar of each plot. From the probabilities, the amplitudes can be calculated although the phase factor may be lost.

Since the code is open source, anyone can customize the walks to their own specific needs, provided they have sufficient background in programming in Python using Qiskit.

Chapter 6

Quantum Walk Noise Distributions

6.1 Motivation

This section explores one application of quantum walks to help prevent data breaches by using an approach called *differential privacy* [33, 31].

Privacy is a growing concern as the amount of data collected from personal devices continues to increase [74]. In particular, medical data is often highly sensitive and, as a result, finding ways to effectively prevent breaches or reduce impact after a breach has already occurred is of great interest [46, 97, 1, 59, 75, 35].

Briefly summarized, differential privacy stems from the belief that privacy cannot be maintained without the injection of noise. It should be noted that few people (if any) require access to an entire dataset. Consequentially, instead of publishing entire datasets, users should be allowed to *query* datasets using aggregate queries that offer general information about a subset of the dataset. For example, 'how many people are above the age of 50?' is an aggregate query. The intention of aggregate queries is to eliminate the querying of any one specific individual to extract that individual's information. It is okay to know that there is one person in the dataset with attribute Z, but we should not know who that person is. However, when query results are accurate, it can be easy to *reconstruct* the dataset or a subset of the dataset. For example, consider a person A queries 'how many people have x (illness/disease)' on a dataset and finds the answer to be 10. Suppose another person B enters their data into the dataset, and that A knows who B is. A then queries the same question and finds the answer to now be 11. If it is likely that no other records in the dataset have changed, A concludes that B has x.

The solution that differential privacy offers is to always return noisy results. For example, if 100 people in a dataset have x, the query could return any number in a nearby range, such as [95,105]. In fact, the true amount of noise injected is often drawn from a distribution, such as the Gaussian or Laplace distributions. The distributions must be adjusted such that not too much nor too little noise is added. Too much noise can render the query result meaningless to those who require accurate results for research purposes, while too little noise results in a higher probability for a successful reconstruction attack.

Since noise is drawn from a distribution using a random variable, the strength of privacy is directly linked to the randomness of the variable drawn. If the noise is not sampled randomly but instead uses pseudo randomness (wherein a predefined sequence is made to appear random), an attacker could replicate the sequence and remove the injected noise. Thus, only true randomness should be used in privacy contexts to minimize the chance of a privacy breach [18].

Unfortunately, sources of true randomness are difficult to create. Many companies and individuals in the 1900s attempted to create sources of randomness, inspired by shuffling of cards, coin tossing, and die rolling, but the resulting randomness tables were often found to have strange biases toward certain numbers [41].

Quantum mechanics states by the uncertainty principle that measuring a quantum particle in superposition yields a truly random result. Thus, through quantum computers, quantum computing can be used as a true source of randomness.

With that in mind, my goal is to create a superposition of states that forms a common distribution. I choose to implement the Gaussian distribution using quantum walks as it can be created using classical random walks, which seems to be a good starting point. Then, upon measurement, the superposition will collapse and return a state at random drawn from the Gaussian distribution.

6.2 Notes on Implementation

While quantum walks can be carried out on any graph-like structure, only quantum walks on a line are examined in this section. The application envisioned – noise sampling for additive mechanisms – returns a single (noisy) real number; it therefore does not make sense to add more dimensions to the walk. However, multidimensional noise sampling can be done with higher dimension quantum walks if desired.

The states on a line are labeled from 0 (at the farthest left point) to n (at the

farthest right point), where n is a power of two. It should be noted that either all even or all odd positions will have a probability of zero dependent on the number of steps the walker takes, assuming the walker begins in only one position (or a superposition of all even or all odd positions). If the walker takes an even number of steps, only even positions will have non-zero probabilities (similarly for odd number of steps). Such functionality must be taken into account when enlisting quantum walks for noise sampling. For this reason, the appropriate even or odd positions (all with zero probability of measurement) have been omitted in Figs. 4.1, 6.1, and 6.2.

6.3 Procedure

Please refer to Appendix C which gives an overview of methodology in the form of lecture slides. These slides provide a high level background on the domain of differential privacy and highlight the challenges encountered when building the quantum walk algorithm for noise sampling.

The quantum walk implementation is largely the same as discussed in Chapter 3. Each walk uses a coin operator and a shift operator, and iterates tossing the coin and taking a step for the number of steps desired. Only one qubit is required to represent q_{dir} since a quantum walk on a line has only two directions. $|0\rangle$ is mapped to the direction 'left', while $|1\rangle$ maps to 'right'. The initial starting position is set to $\frac{n}{2}$ given a total of n positions (when $n = 2^d$, $d \in \mathbb{Z}^+$). For ease of notation, the q_{dir} register will be denoted with subscript d, and the q_{pos} register with subscript p.

6.3.1 Hadamard Walk

The Hadamard walk is a basic walk in which the coin operator $C = H$ (where H is the Hadamard gate). The Hadamard gate introduces equal-amplitude superposition, so when it is applied to the direction qubit, it sets the direction to be a superposition of both left and right. It may be tempting to conclude that if we are travelling both left and right equally, we should achieve the Gaussian distribution as a classical random walk would. However, due to destructive interference introduced by the Hadamard coin, the distribution we see is as in Fig. 4.1. Because $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, repeated applications of the Hadamard coin results in both negative and positive terms of the same state, causing such states to sum to 0.

6.3.2 Gaussian Distribution Walk

Since only $H|1\rangle$ introduces negative terms, we can eliminate destructive interference if we never apply the Hadamard coin to $|1\rangle$. In order to achieve that, we must take the output of applying the Hadamard coin once $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and change it back to $|0\rangle$. Applying a second Hadamard gate will bring us back from the super-positional state to $|0\rangle$ but only when remaining part of the state (i.e., $|pos\rangle$) is the same. Unfortunately, we cannot assume that this will always be the case.

For example, consider even the first step of the walk with four total positions starting at $|10\rangle$. The initial state is $|0\rangle_d \otimes |10\rangle_p$. We apply the Hadamard coin q_{dir} to yield $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_d \otimes |10\rangle_p$. After the shift operator is applied, we obtain $\frac{1}{\sqrt{2}}(|0\rangle_d \otimes |01\rangle_p + |1\rangle_d \otimes |11\rangle_p)$. Note that the position qubits are not the same state; thus, applying another Hadamard gate will not bring the direction qubit back to $|0\rangle$ but instead introduce a negative term when applied to $|1\rangle$ (eq. (6.1)) below. Here, H_d represents applying the H gate upon q_{dir} only.

$$H_d(\frac{1}{\sqrt{2}}(|0\rangle_d \otimes |01\rangle_p + |1\rangle_d \otimes |11\rangle_p)) \\ = \frac{1}{2}(|0\rangle_d \otimes |01\rangle_p + |1\rangle_d \otimes |01\rangle_p \\ + |0\rangle_d \otimes |11\rangle_p - |1\rangle_d \otimes |11\rangle_p) \quad (6.1)$$

Another solution is to keep the $|1\rangle$ direction terms, but remove the negative phase added by the Hadamard gate. The Pauli Z gate can be used to multiply a phase by -1 so long as the qubit is set to $|1\rangle$ when the gate is applied. As a refresher from Chapter 2, in Dirac notation, $Z|1\rangle = -|1\rangle$, while $Z|0\rangle = |0\rangle$. However, we cannot apply the Z gate directly to the direction qubit as it would multiply all $|1\rangle$ terms by -1, causing the initially negative phases to become positive at the cost of making the initially positive phases become negative (eq. (6.2)). The Z gate would therefore require a controlled application where the control necessitates knowledge of which terms were $|1\rangle$ before the Hadamard gate was applied to them (on the previous iteration). More explicitly, if q_{dir} was $|1\rangle$ before the Hadamard gate was applied, only then should one apply the controlled Z gate to reverse the negative $|1\rangle$ term introduced. Thus, the original values of q_{dir} would have to be stored somewhere or copied, but it is known in quantum mechanics that states cannot be cloned due to

the no-cloning theorem [109], making this a very difficult task.

$$
\begin{aligned}
Z_d(\frac{1}{2}(|0\rangle_d \otimes |01\rangle_p + |1\rangle_d \otimes |01\rangle_p \\
&+ |0\rangle_d \otimes |11\rangle_p - |1\rangle_d \otimes |11\rangle_p)) \\
&= \frac{1}{2}(|0\rangle_d \otimes |01\rangle_p - |1\rangle_d \otimes |01\rangle_p \\
&\qquad + |0\rangle_d \otimes |11\rangle_p + |1\rangle_d \otimes |11\rangle_p) \quad (6.2)
\end{aligned}
$$

Instead, auxilliary qubits were used as a solution to eliminate destructive interference. In each iteration of the walk after the Hadamard gate was applied to the q_{dir}, the resulting qubit in superposition was swapped with a qubit set to $|0\rangle$. This is the same solution as discussed in quantum walks search algorithm implementation in Chapter 3. The effect is that no negative phases can be introduced because each qubit has no more than one Hadamard gate applied to it; the cost is the walk now has added linear space complexity $O(t)$ where t is the number of steps. The Gaussian distribution can be created for nearly any mean and standard deviation with limitation only due to the current size of accessible quantum computers (cf. Fig. 6.1). To change the mean, the initial position of the walker is changed. To adjust the standard deviation, the number of steps is changed, where more steps result in higher standard deviation. Multiple walkers can be added to create customizable distibutions with 'multiple means' (cf. Fig. 6.2).

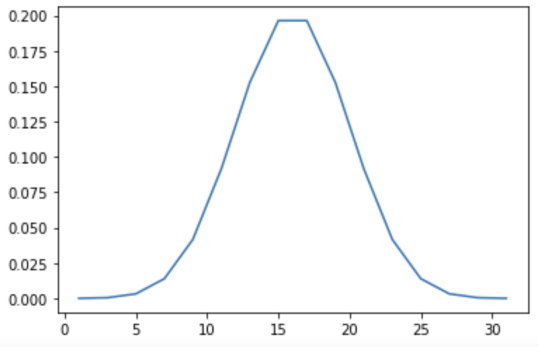


Figure 6.1: The Gaussian distribution created by a quantum walk with 2^5 positions and 15 steps.

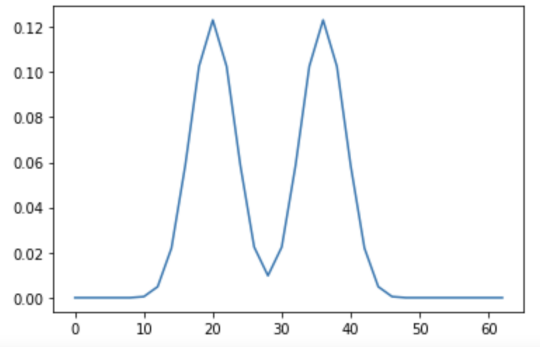


Figure 6.2: A Gaussian distribution with two walkers using 2^6 positions and 10 steps.

6.3.3 Equal Distribution

A useful distribution is the equal distribution, wherein each element has an equal probability of being selected at random. The equal distribution is perhaps the easiest distribution to create using quantum computing since the Hadamard gate serves exactly that purpose. Creating the equal distribution using a quantum walk requires no steps to be taken and thus can be performed in $O(1)$ time (cf. Fig. 6.3).

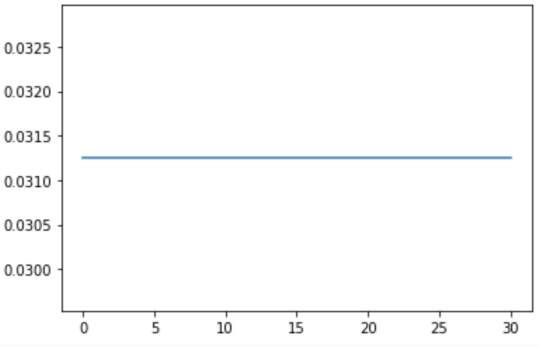


Figure 6.3: The equal distribution on 2^5 positions.

Chapter 7

Conclusions and Future Work

7.1 Contributions

Although the full potential of quantum walks is still being explored, quantum walks algorithms appear to be a promising vein of research with applications in search algorithms [23, 10], decoherence [2], reinforcement learning [76, 101], encryption [66], image segmentation [60], and many other fields. However, implementations of quantum walks in 2023 are mostly carried out at the hardware level [91, 73, 100, 87]. As a result, specialized physical equipment is required to replicate such implementations, and the algorithms therefore may be difficult to translate into the quantum universal gate set algorithms.

High level algorithms for one-, two-, and three-dimensional quantum walks are presented in Chapter 3, with code publicly available via a GitHub repository whose link is accessible in Chapter 4. These algorithms are specifically designed to be simple and employ only basic quantum gates: X, Z, Hadamard, and Toffoli gates, allowing beginners in quantum computing to more easily understand how each component works. Since only basic quantum gates are used, the code can be run on any backend which accepts the universal gate set.

Novel quantum walk visualizations can be plotted using the QWalkVis website application as outlined in Chapter 4. These visualizations plot not only the probability associated with each position the walker is in, but also the shape of the walk itself in one-, two-, and three-dimensional space. Since the plots are generated through fields and buttons on a website, no knowledge of programming is necessary to make them.

Lesson plans and materials for teaching superposition, interference, and measure-

ment using quantum walks are presented in Chapter 5. With the QWalkVis tool, it is easy to create custom visualizations for superposition and interference that can help visual learners to learn those concepts. The visualizations can also be used in research or to generate interest 'at-a-glance' for quantum walks.

Lastly, a specific potential application for quantum walks is discussed in Chapter 6. Adding noise via additive mechanisms in order to preserve privacy in data relies on true randomness. Since measurement in quantum mechanics is a source of true randomness, using a quantum algorithm could be of great benefit. A modified version of the quantum walk algorithm is capable of creating a Gaussian distribution which, when measured, will collapse to a random single discrete state according to the Gaussian distribution. If the distribution is mapped to a chosen amount of noise, the quantum walk algorithm acts as a quantum additive mechanism.

7.2 Future Work

7.2.1 Walks on Other Spaces

The quantum walk implementations in this thesis are currently only for walks that take place on lines, grids, or cubes. Other shapes, especially spaces that are not necessarily symmetrical, such as graphs, cannot be traversed with the shift operator introduced in Chapter 3. Shapes whose dimensions cannot be represented efficiently in binary also cannot be traversed using this shift operator, as the operator takes advantage of properties specific to the binary numbering system.

Graph algorithms are of great interest as many computing problems can be translated into graph problems (e.g., independent set, vertex cover, minimum and maximum flow networks). Thus, a quantum walk algorithm that could traverse any arbitrary graph is highly sought after. There exist quantum walk algorithms which operate on complete graphs [29], but when vertices in a graph are permitted to have varying degrees, complex coin and shift operators are required.

On non-complete graphs, mapping of directions must be amended, since up, down, left, and right can be arbitrarily chosen for any graph. Furthermore, one can 'drag' vertices and order them in any fashion, so any edge can be drawn in any way and along any axis. To solve this, all edges ideally would be labelled, and an additional function $o(e, v)$ created, where e is an edge, v is a vertex, and $o(e, v)$ returns the endpoint of edge e that is not v (the 'other' endpoint).

The coin operator would also need to identify all adjacent edges, $adj(v)$, of a given vertex, v, and then construct an equal superposition from $adj(v)$. When $|adj(v)|$ is not a power of two, this becomes more challenging using qubits, although there exist algorithms that can create superpositions when $|adj(v)|$ not only is not a power of two, but also an odd number [93, 89].

The shift operator would then take in a superposition of positions and edges and take a step along each edge. Since different vertices will have different adjacent edges, the shift operator need somehow be able to map each edge to its adjacent vertex.

7.2.2 Quantum Walks Visualizations and QWalkVis

Much can be added to the visualizations of the quantum walks themselves. By adding more customizable parameters to QWalkVis, different visualizations can be created. For example, it could be optional to scale the probabilities on the colourbar. The initial starting position of the walk could be selected freely. The number of walkers could also be increased or decreased. Walking off of one side and appearing back on the other (torus functionality) could be optional.

It may be best to start with a survey, however. Both researchers and those new to quantum walks could be asked what features they would like to see added to QWalkVis.

It should also be noted that since QWalkVis is open source and available on GitHub, anyone can download and change the code to fit their own needs.

7.2.3 Visualizations in Quantum Computing

In general, more visualizations in quantum computing would provide more content for visual learners. Famous algorithms such as Grover's algorithm (amplitude amplification in particular) could benefit from more diverse visualizations beyond circuit diagrams. New diagrams for representing qubits, superposition, entanglement, and many more concepts may provide more points of view and considerations for both new and established learners.

Bibliography

[1] Tanvir Alam and Mohammad S. Rahman. To trace or not to trace: Saving lives from covid-19 at the cost of privacy breach in bangladesh. *Qatar Medical Journal*, 3:35, 2020.

[2] Andrea Alberti, Wolfgang Alt, Reinhard Werner, and Dieter Meschede. Decoherence models for discrete-time quantum walks and their application to neutral atom experiments. *New Journal of Physics*, 16(12):123052, Dec 2014.

[3] Catalina Albornoz. Xanadu quantum codebook. `https://codebook.xanadu.ai`, 2023. Accessed: 2023-08-21.

[4] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor oracle: A new structure for pattern matching. In Jan Pavelka, Gerard Tel, and Miroslav Bartošek, editors, *SOFSEM'99: Theory and Practice of Informatics*, pages 295–310, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[5] Osamah Alrouwab. Evaluating efficiency of some exact string-matching algorithms on large-scale genome. *American Journal of Computer Science and Information Technology*, 9(9):112, 2021.

[6] Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1(04):507–518, 2003.

[7] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

[8] Andris Ambainis, Eric Bach, Ashwin Nayak, Ashvin Vishwanath, and John Watrous. One-dimensional quantum walks. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, page 37–49, New York, NY, USA, 2001.

[9] Andris Ambainis, András Gilyén, Stacey Jeffery, and Martins Kokainis. Quadratic speedup for finding marked vertices by quantum walks. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 412–424, New York, NY, USA, 2020.

[10] Andris Ambainis, Julia Kempe, and Alexander Rivosh. Coins make quantum walks faster. *arXiv:quant-ph/0402107*, 2004.

[11] Prashanti P. Angara. From qubits to quantum teleportation. `https://pangara.github.io/q2qt/`, 2020. Accessed: 2023-08-21.

[12] Ryo Asaka, Kazumitsu Sakai, and Ryoko Yahagi. Quantum random access memory via quantum walk. *Quantum Science and Technology*, 6(3):035004, 2021.

[13] Gérard Assayag and Shlomo Dubnov. Using factor oracles for machine improvisation. *Soft Computing*, 8(9):604–610, 2004.

[14] Christiane Baadte and Bozana Meinhardt-Injac. The picture superiority effect in associative memory: A developmental study. *British Journal of Developmental Psychology*, 37(3):382–395, 2019.

[15] Paul Benioff. Space searches with a quantum robot. *arXiv:quant-ph/0003006*, 2001.

[16] Chris Bernhardt. *Quantum Computing for Everyone*. Mit Press, 2019.

[17] Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes. In *28th Annual IEEE Symposium on Foundations of Computer Science (sfcs 1987)*, pages 118–126, 1987.

[18] Carl Bosley and Yevgeniy Dodis. Does privacy require true randomness? In Salil P. Vadhan, editor, *Theory of Cryptography*, pages 1–20, Berlin, Heidelberg, 2007.

[19] Dirk Bouwmeester, Irene Marzoli, Gerwin P Karman, Wolfgang Schleich, and Johannes P Woerdman. Optical galton board. *Physical Review A*, 61(1):013410, 1999.

[20] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: Practical asynchronous byzantine agreement using cryptography (extended abstract). Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing '00, page 123–132, New York, NY, USA, 2000.

[21] Andrew M. Childs. Universal computation by quantum walk. *Phys. Rev. Lett.*, 102:180501, May 2009.

[22] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 59–68, New York, NY, USA, 2003.

[23] Andrew M. Childs and Jeffrey Goldstone. Spatial search by quantum walk. *Physical Review A*, 70(2):022314, 2004.

[24] Google Cirq. Quantum walk. `https://quantumai.google/cirq/experiments/quantum_walks`, 2023. Accessed: 2023-07-30.

[25] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[26] Don Coppersmith. An approximate fourier transform useful in quantum factoring. *arXiv:quant-ph/0201067*, 2002.

[27] Larry Cuban. *Teachers and Machines: The Classroom Use of Technology Since 1920*. Teachers College Press, 1986.

[28] Alexandre S. de Abreu, Luís F. I. Cunha, Celina M. H. de Figueiredo, Franklin L. Marquezino, Daniel F. D. Posner, and Renato Portugal. Total tessellation cover and quantum walk. *arXiv:2002.08992*, 2020.

[29] Luciano S. de Souza, Jonathan H. A. de Carvalho, and Tiago A. E. Ferreira. Lackadaisical quantum walk in the hypercube to search for multiple marked vertices. In *Intelligent Systems: 10th Brazilian Conference, BRACIS 2021, Virtual Event, November 29 – December 3, 2021, Proceedings, Part I*, page 249–263, Berlin, Heidelberg, 2021.

[30] Cirq Developers. Cirq. Jul 2023. Refer to full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors.

[31] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03, page 202–210, New York, NY, USA, 2003.

[32] Paul A. M. Dirac. A new notation for quantum mechanics. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 35, pages 416–418. Cambridge University Press, 1939.

[33] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 1–12, Berlin, Heidelberg, 2006.

[34] Quantum Flytrap. Quantum flytrap virtual lab. `https://lab.quantumflytrap.com/lab?mode=waves`, 2023. Accessed: 2023-01-24.

[35] Shadan Ghaffaripour and Ali Miri. A decentralized, privacy-preserving and crowdsourcing-based approach to medical research. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 4510–4515, 2020.

[36] Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006.

[37] Temple Grandin. *Thinking in Pictures, Expanded Edition: My Life with Autism*. Vintage, 2008.

[38] Quantum Computing Group. Hiperwalk. `http://qubit.lncc.br/qwalk/`, 2023. Accessed: 2023-07-30.

[39] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996.

[40] Aram Harrow. Learning quantum computing. `https://www.mit.edu/~aram/advice/quantum.html`, 2023. Accessed: 2023-08-21.

[41] Brian Hayes. Computing science: Randomness as a resource. *American Scientist*, 89(4):300–304, 2001.

[42] Mary Hegarty. Dynamic visualizations and learning: Getting to the difficult questions. *Learning and Instruction*, 14(3):343–351, 2004.

[43] Denis R. Hirschfeldt, André Nies, and Frank Stephan. Using random sets as oracles. *Journal of the London Mathematical Society*, 75(3):610–622, 07 2007.

[44] Tad Hogg. Quantum random walk. `http://demonstrations.wolfram.com/QuantumRandomWalk/`, 2023. Accessed: 2023-07-30.

[45] IBM. Composer user guide. `https://learning.quantum-computing.ibm.com/tutorial/composer-user-guide`, 2023. Accessed: 2023-08-21.

[46] John Jackson. The costs of medical privacy breach. *MD Advisor*, 8(3):4–12, 2015.

[47] Stacey Jeffery and Sebastian Zur. Multidimensional quantum walks. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 1125–1130, New York, NY, USA, 2023.

[48] Jodie Jenkinson. Molecular biology meets the learning sciences: Visualizations in education and outreach. *Journal of Molecular Biology*, 430(21):4013–4027, 2018.

[49] Yao-Yao Jiang, Peng-Cheng Chu, Wen-Bin Zhang, and Hong-Yang Ma. Quantum walk search algorithm for multi-objective searching with iteration autocontrolling on hypercube. *Chinese Physics B*, 31(4):040307, 2022.

[50] Zhen-Ming Jiang, Ahmed E. Hassan, and Richard C. Holt. Visualizing clone cohesion and coupling. In *2006 13th IEEE Asia Pacific Software Engineering Conference (APSEC'06)*, pages 467–476, 2006.

[51] Kevin Judd. Galton's quincunx: Random walk or chaos? *International Journal of Bifurcation and Chaos*, 17(12):4463–4469, 2007.

[52] Viv Kendon. Decoherence in quantum walks—a review. *Mathematical Structures in Computer Science*, 17(6):1169–1220, 2007.

[53] Viv Kendon. How to compute using quantum walks. *arXiv:2004.01329*, 2020.

[54] Viv Kendon and Ben Tregenna. Decoherence can be useful in quantum walks. *Phys. Rev. A*, 67:042315, Apr 2003.

[55] Jalil. Khatibi Moqadam and Ali T. Rezakhani. Boundary-induced coherence in the staggered quantum walk on different topologies. *Phys. Rev. A*, 98:012123, Jul 2018.

[56] Alexei Y. Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv:quant-ph/9511026*, 1995.

[57] Martins Kokainis, Krišjānis Prūsis, Jevgenijs Vihrovs, Vyacheslavs Kashcheyevs, and Andris Ambainis. Strong dispersion property for the quantum walk on the hypercube. *Journal of Physics A: Mathematical and Theoretical*, 55(49):495301, Dec 2022.

[58] Mojtaba Komeili. Quantum walk on two-dimensional torus and grid. *Quantum*, 2018:05–28, 2018.

[59] Shannon K. S. Kroes, Mart P. Janssen, Rolf H. H. Groenwold, and Matthijs van Leeuwen. Evaluating privacy of individuals in medical data. *Health Informatics Journal*, 27(2):1460458220983398, 2021. PMID: 34075842.

[60] Michał Krok, Katarzyna Rycerz, and Marian Bubak. Application of continuous time quantum walks to image segmentation. In *Computational Science – ICCS 2019: 19th International Conference, Faro, Portugal, June 12–14, 2019, Proceedings, Part II*, page 17–30, Berlin, Heidelberg, 2019. Springer-Verlag.

[61] Maithilee Kunda and Ashok K. Goel. Thinking in pictures as a cognitive account of autism. *Journal of Autism and Developmental Disorders*, 41(9):1157–1177, 2011.

[62] Arnaud Lefebvre and Thierry Lecroq. Compror: on-line lossless data compression with a factor oracle. *Information Processing Letters*, 83(1):1–6, 2002.

[63] Guanzhong Li and Lvzhou Li. Optimal exact quantum algorithm for the promised element distinctness problem. *arXiv:2211.05443*, 2022.

[64] Xi Li, Hanwu Chen, Yue Ruan, Zhihao Liu, and Wenjie Liu. Continuous-time quantum walks on strongly regular graphs with loops and its application to

spatial search for multiple marked vertices. *Quantum Information Processing*, 18:1–20, 2019.

[65] Xi Li, Mingyou Wu, Hanwu Chen, and Zhibao Liu. Algorithm for finding the maximum clique based on continuous time quantum walk. *arXiv:1912.02728*, 2021.

[66] Pai Liu, Shihua Zhou, and Wei Qi Yan. A 3d cuboid image encryption algorithm based on controlled alternate quantum walk of message coding. *Mathematics*, 10(23), 2022.

[67] Kai Lu, Yi Zhang, Kai Xu, Yinghui Gao, and Richard C. Wilson. Approximate maximum common sub-graph isomorphism based on discrete-time quantum walk. In *2014 22nd International Conference on Pattern Recognition*, pages 1413–1418, 2014.

[68] Zhou Lu, Nataly Brukhim, Paula Gradu, and Elad Hazan. Projection-free adaptive regret with membership oracles. In *International Conference on Algorithmic Learning Theory*, pages 1055–1073. PMLR, 2023.

[69] Satya N. Majumdar, Sanjib Sabhapandit, and Grégory Schehr. Random walk with random resetting to the maximum position. *Physical Review E*, 92(5):052126, 2015.

[70] Amrita Mandal, Rohit S. Sarkar, and Bibhas Adhikari. Localization of two dimensional quantum walks defined by generalized grover coins. *Journal of Physics A: Mathematical and Theoretical*, 56(2):025303, Jan 2023.

[71] Samuel Marsh and Jingbo Wang. A quantum walk assisted approximate algorithm for bounded np optimisation problems. *arXiv:1804.08227*, 2018.

[72] Inc. Meta Platforms. React. `https://reactjs.org`, 2023. Accessed: 2023-02-28.

[73] Yusuke Mizutani, Tomoyuki Horikiri, Leo Matsuoka, Yusuke Higuchi, and Etsuo Segawa. Implementation of a discrete-time quantum walk with a circulant matrix on a graph by optical polarizing elements. *Phys. Rev. A*, 106:022402, Aug 2022.

[74] Tarek Moulahi. Joining Formal Concept Analysis to Feature Extraction for Data Pruning in Cloud of Things. *The Computer Journal*, 65(9):2484–2492, 06 2021.

[75] Surma Mukhopadhyay, Ramsankar Basak, and Brian J. Reithel. Security/privacy perceptions in patient use of online medical records: Study from a large national survey. *International Journal of Healthcare Information Systems and Informatics (IJHISI)*, 16(4):1–18, 2021.

[76] Thomas Mullor, David Vigouroux, and Louis Béthune. Efficient circuit implementation for coined quantum walks on binary trees and application to reinforcement learning. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 436–443, Los Alamitos, CA, USA, Dec 2022.

[77] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[78] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1999.

[79] Pallets. Flask. `https://flask.palletsprojects.com/en/2.2.x/#api-reference`, 2023. Accessed: 2023-02-28.

[80] Dennis K. Peters and David L. Parnas. Using test oracles generated from program documentation. *IEEE Transactions on Software Engineering*, 24(3):161–173, 1998.

[81] Rod Pierce. Quincunx. `https://www.mathsisfun.com/data/quincunx.html`, 2023. Accessed: 2023-07-30.

[82] David N. Rapp. Mental models: Theoretical issues for visualizations in science education. In *Visualization in Science Education*, pages 43–60. Springer, 2005.

[83] Daniel Reitzner, Daniel Nagaj, and Vladimir Buzek. Quantum walks. *arXiv:1207.7283*, 2012.

[84] Mason L. Rhodes and Thomas G. Wong. Search on vertex-transitive graphs by lackadaisical quantum walk. *Quantum Information Processing*, 19(9), 2020.

[85] Debra J. Richardson, Stephanie L. Aha, and Owen O'Malley. Specification-based test oracles for reactive systems. In *Proceedings of the 14th International Conference on Software Engineering*, ICSE '92, page 105–118, New York, NY, USA, 1992.

[86] Amit Saha, Sudhindu B. Mandal, Debasri Saha, and Amlan Chakrabarti. One-dimensional lazy quantum walk in ternary system. *IEEE Transactions on Quantum Engineering*, 2:1–12, 2021.

[87] Barry C. Sanders, Stephen D. Bartlett, Ben Tregenna, and Peter L. Knight. Quantum quincunx in cavity quantum electrodynamics. *Physical Review A*, 67(4):042305, 2003.

[88] The Coding School. Qubitxqubit. `https://www.qubitbyqubit.org`, 2023. Accessed: 2023-08-21.

[89] Risa Segawa and Shigeru Yamashita. An efficient generation of arbitrary superposition of basis states. *The 20th Asian Quantum Information Science Conference*, AQIS2020:211, 2020.

[90] Neeraj Sharma and Lalit M. Aggarwal. Automated medical image segmentation techniques. *Journal of Medical Physics/Association of Medical Physicists of India*, 35(1):3, 2010.

[91] Zi-Yu Shi, Hao Tang, Zhen Feng, Yao Wang, Zhan-Ming Li, Jun Gao, Yi-Jun Chang, Tian-Yu Wang, Jian-Peng Dou, Zhe-Yong Zhang, Zhi-Qiang Jiao, Wen-Hao Zhou, and Xian-Min Jin. Quantum fast hitting on glued trees mapped on a photonic chip. *Optica*, 7(6):613–618, Jun 2020.

[92] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[93] Alok Shukla and Prakash Vedula. An efficient quantum algorithm for preparation of uniform quantum superposition states. *arXiv:2306.11747*, 2023.

[94] Ilya Sinayskiy and Francesco Petruccione. Efficiency of open quantum walk implementation of dissipative quantum computing algorithms. *Quantum Information Processing*, 11:1301–1309, 2012.

[95] Matt Staats, Michael W. Whalen, and Mats P. E. Heimdahl. Programs, tests, and oracles: The foundations of testing revisited. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, page 391–400, New York, NY, USA, 2011.

[96] Robert S. Sutor. *Dancing with Qubits: How Quantum Computing Works and How It Can Change the World*. Packt Publishing Ltd, 2019.

[97] Shipra Varshney, Dheeraj Munjal, Orijit Bhattacharya, Shagun Saboo, and Nikunj Aggarwal. Big data privacy breach prevention strategies. In *2020 IEEE International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC)*, pages 1–6, 2020.

[98] Salvador E. Venegas-Andraca. Quantum walks: a comprehensive review. *Quantum Information Processing*, 11(5):1015–1106, 2012.

[99] Ce Wang. The uniform measure for quantum walk on hypercube: A quantum Bernoulli noises approach. *Journal of Mathematical Physics*, 63(11):113501, 11 2022.

[100] Kunkun Wang, Yuhao Shi, Lei Xiao, Jingbo Wang, Yogesh N. Joglekar, and Peng Xue. Experimental realization of continuous-time quantum walks on directed graphs and their application in pagerank. *Optica*, 7(11):1524–1530, Nov 2020.

[101] Yao Wang, Zi-Wei Cui, Yong-Heng Lu, Xiao-Ming Zhang, Jun Gao, Yi-Jun Chang, Man-Hong Yung, and Xian-Min Jin. Integrated quantum-walk structure and nand tree on a photonic chip. *Phys. Rev. Lett.*, 125:160502, Oct 2020.

[102] Yao Wang, Zi-Wei Cui, Yong-Heng Lu, Xiao-Ming Zhang, Jun Gao, Yi-Jun Chang, Man-Hong Yung, and Xian-Min Jin. Integrated quantum-walk structure and nand tree on a photonic chip. *Phys. Rev. Lett.*, 125:160502, Oct 2020.

[103] Allan Wing Bocanegra and Salvador Venegas-Andraca. Circuit implementation of discrete-time quantum walks via the shunt decomposition method. *Quantum Information Processing*, 22, 03 2023.

[104] Thomas G. Wong. Faster search by lackadaisical quantum walk. *Quantum Information Processing*, 17(3):1–9, 2018.

[105] Xiaodi Wu. Mini-library on quantum information and computation. `https://www.cs.umd.edu/~xwu/mini_lib.html`, 2023. Accessed: 2023-08-21.

[106] Xanadu. Learn quantum programming. `https://pennylane.ai/qml/`, 2023. Accessed: 2023-08-21.

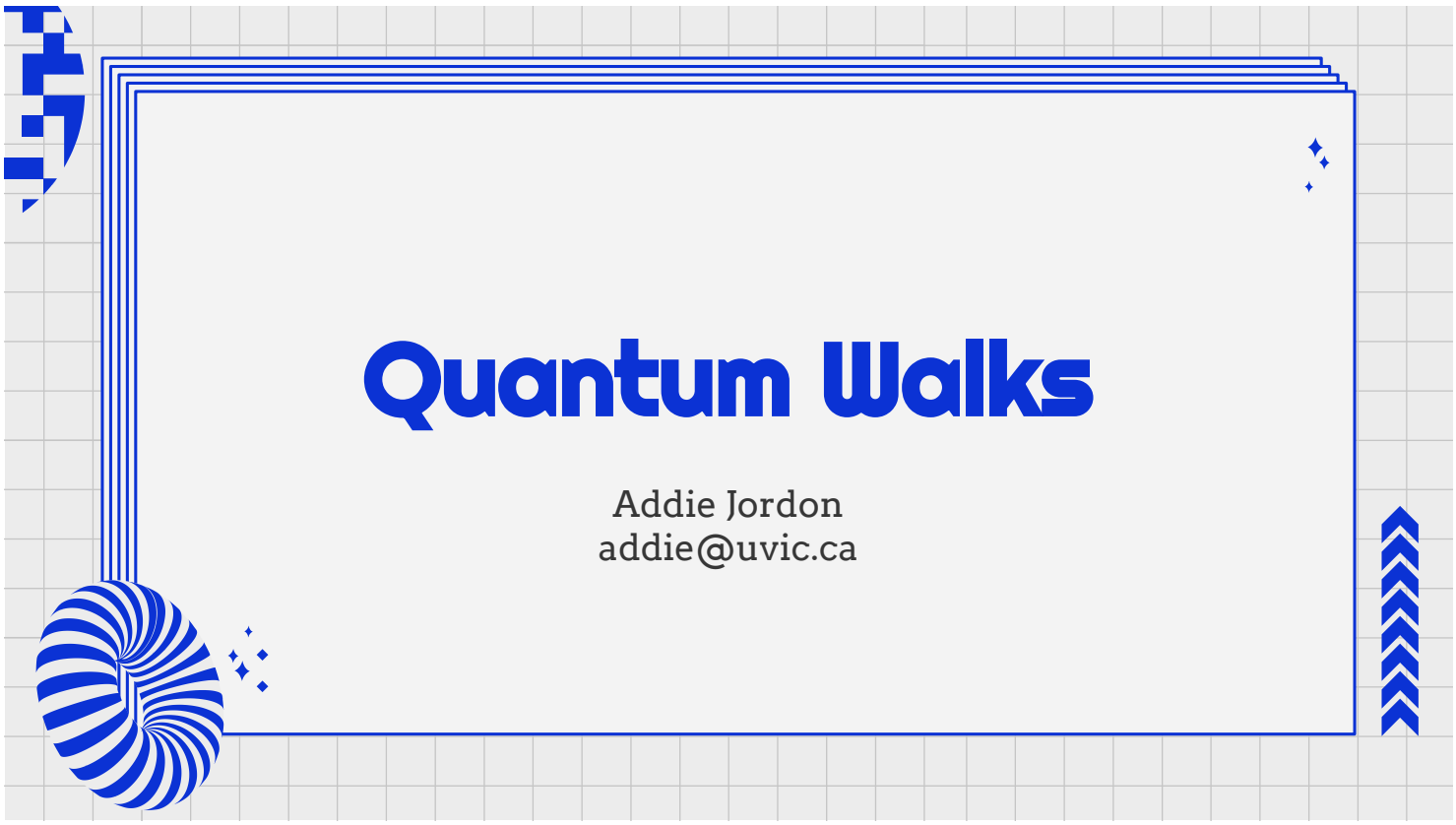
[107] Feng Xia, Jiaying Liu, Hansong Nie, Yonghao Fu, Liangtian Wan, and Xiangjie Kong. Random walks: A review of algorithms and applications. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(2):95–107, 2019.

[108] Yongzhen Xu, Delong Zhang, and Lvzhou Li. Robust quantum walk search without knowing the number of marked vertices. *Phys. Rev. A*, 106:052207, Nov 2022.

[109] Muhammad S. Zubairy. *Quantum Mechanics for Beginners*. Oxford University Press, 2020.

Appendices

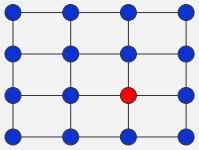
A Sample Lecture Slides for Introducing Quantum Walks



Motivation

Problem definition

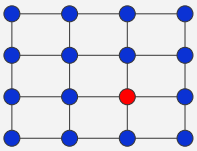
Quantum Walks can be used as a search algorithm.



This begs the question, why not just use Grover's search?

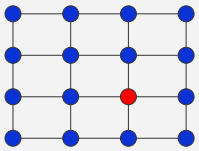
Motivation: grid search

Consider N elements arranged in an n x n grid, $n^2 = N$.



Motivation: grid search

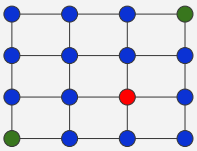
At each step, we can query the current cell or move to an adjacent cell.



Motivation: grid search

Grover's algorithm solves this problem in O(sqrt(N)) queries.

But! Requires O(sqrt(N)) moves between each query!

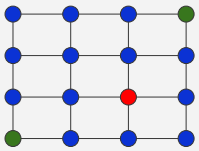


Total time is O(N)!

Solution: Quantum walks

Motivation: grid search

Conclusion: Quantum walks are good when dealing with a structured search space.



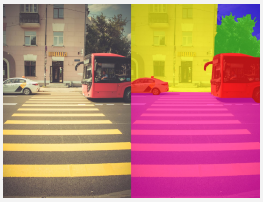
"Sometimes, we have a search space, where after testing an item, it is faster to test a neighboring item than an arbitrary item."

—A. Ambainis

A. Ambainis, "Quantum search algorithms," *ACM SIGACT News*, pp. 22-35, 2004. [Online]. Available: https://www.researchgate.net/profile/Andris-Ambainis/publication/234765968_Quantum_search_algorithms/links/09e41512c3dd9fbf59000000/Quantum-search-algorithms.pdf

Other (Possible) Applications

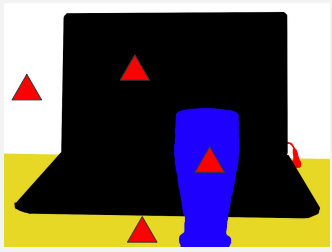
Quantum Walks can be used for image segmentation



Classical random walks are currently being used for image segmentation.

Other (Possible) Applications

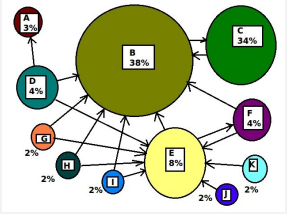
Image segmentation



- user picks seed locations
- walks performed on each pixel to see which seed is closest

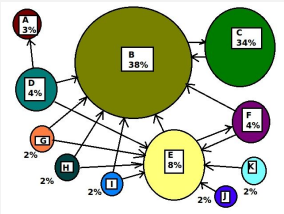
Other (Possible) Applications

Quantum Walks can be used for RageRank.



Classical random walks are currently being used for PageRank.

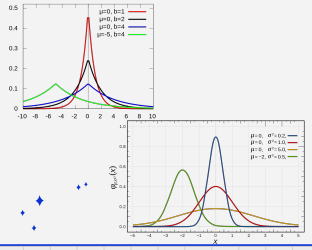
Other (Possible) Applications



- pages are more important when they have multiple pages pointing to them
- web represented as a graph
- probability distribution of random walks

Other (Possible) Applications

Privacy: noise distributions and noise sampling.



Quantum measurement is one of the very few natural sources of true randomness.

Other (Possible) Applications

Privacy: noise distributions and noise sampling.

True randomness is very difficult to find!

Currently, most applications use pseudorandomness.

Other (Possible) Applications

Privacy: noise distributions and noise sampling.

Pseudorandomness:
- good enough?

Other (Possible) Applications

Privacy: noise distributions and noise sampling.

Pseudorandomness:
- good enough?

Name	Age	Address	Medical condition
Max Mustermann	58	123 Musterstraße	x
Mahmoud Ibrahim	23	123 Mahrousa St	x

Other (Possible) Applications

Conclusion: **most** people do not need access to individual records.

Name	Age	Address	Medical condition
Max Mustermann	58	123 Musterstraße	x
Mahmoud Ibrahim	23	123 Mahrousa St	x

Aggregate data is just as useful and much more safe.

But: can easily solve accurate aggregate data!

Other (Possible) Applications

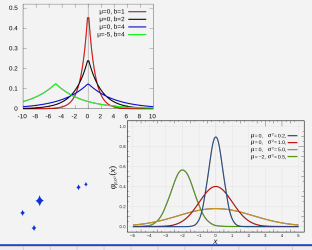
Name	Age	Address	Medical condition
Max Mustermann	58	123 Musterstraße	x
Mahmoud Ibrahim	23	123 Mahrousa St	x

Reconstruction attack: If we receive highly accurate answers, we can use linear programming to reconstruct databases.

Solution? Differential privacy.

Other (Possible) Applications

Differential privacy purposefully returns inaccurate data.



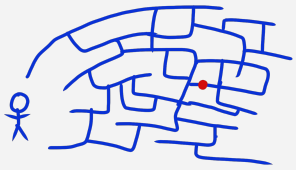
The amount of inaccuracy (noise) is usually drawn from a distribution.

Quantum walks can be used to create noise distributions!

How do quantum walks work?

The Problem

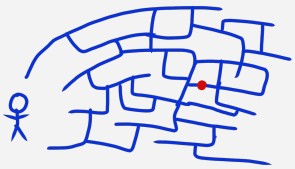
Consider you are in a new city and trying to find your way home. You are completely lost. You have a coin.



The Solution: Random walk

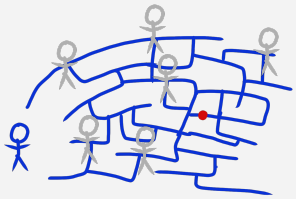
Flip the coin to decide which direction you go.

- heads = go left
- tails = go right



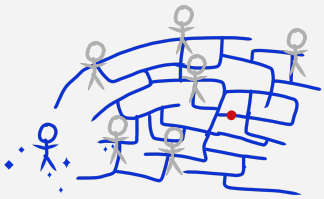
The Solution: Quantum walk

Now imagine both you and the coin can exist in superposition!



The Solution: Quantum walk

Now imagine both you and the coin can exist in superposition!



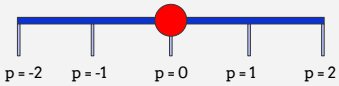
You will need two Hilbert spaces:

- coin space (direction)
- position space

and two operators:

- coin operator
- shift operator

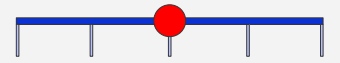
Classical random walk



- C is the "coin operator"
  - coin can be L or R
- S is the "shift operator"
  - move L or R based on coin

- let L = 0 and R = 1

Quantum walk



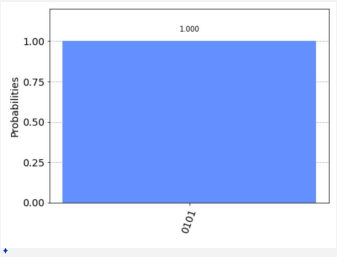
- C is the "coin operator"
  - coin can be |0> or |1> or a superposition of both
- S is the "shift operator"
  - move L or R based on coin
- Can exist in a superposition of positions

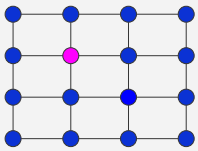
Quantum walk

Now we have defined a way to move around the line using quantum notation.

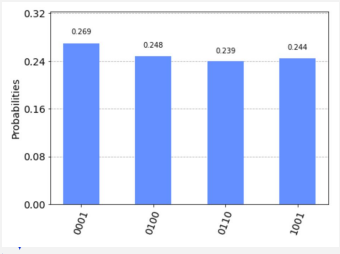
Let's visualize it!

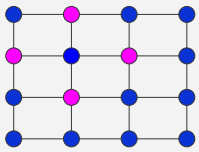
Walk on a Grid (Torus)



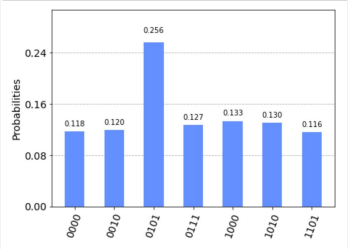


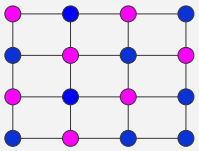
Walk on a Grid (Torus)



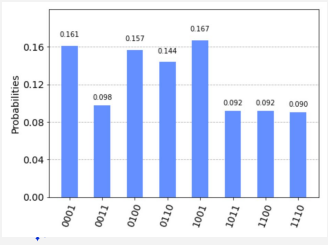


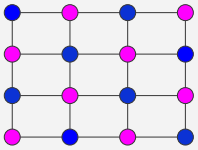
Walk on a Grid (Torus)



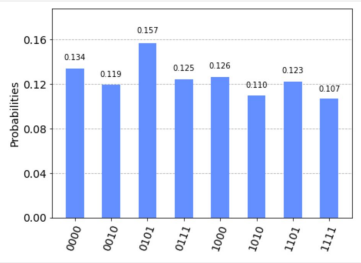


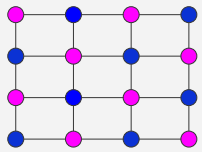
Walk on a Grid (Torus)





Walk on a Grid (Torus)



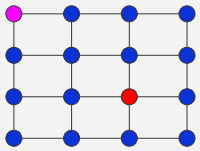


Not super useful, but shows us how it works!

Mods

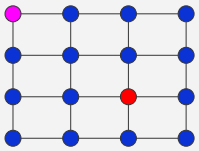
Searching

- idea: once we reach the target state, we stay there.



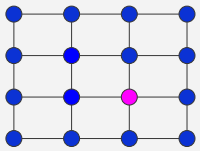
Searching

- with one walker (classically):



Searching

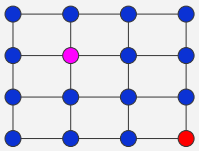
- this took 10 steps.



(Since this is a random algorithm, the worst case running time is technically infinity, although probability approaches 1 as number of steps approach infinity.)

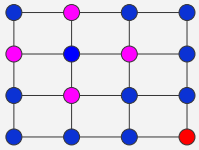
Searching quantumly

- Let's start in one state, but go in all four directions at each step.



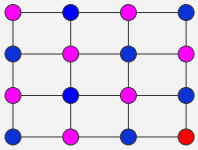
Searching quantumly

- Let's start in one state, but go in all four directions at each step.



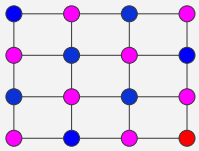
Searching quantumly

- Let's start in one state, but go in all four directions at each step.



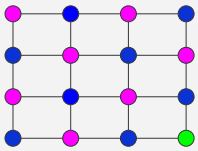
Searching quantumly

- Let's start in one state, but go in all four directions at each step.



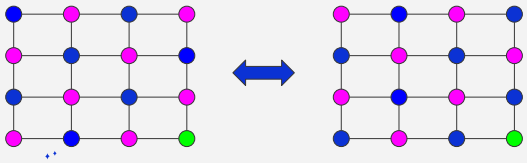
Searching quantumly

- Let's start in one state, but go in all four directions at each step.



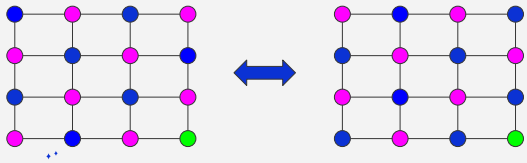
Searching quantumly

- Maybe you can convince yourself that these two states simply repeat over and over again!



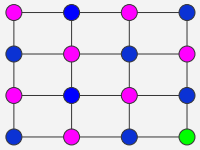
Searching quantumly

- We took 4-5 steps.
  - the same number of steps is required no matter where you start.

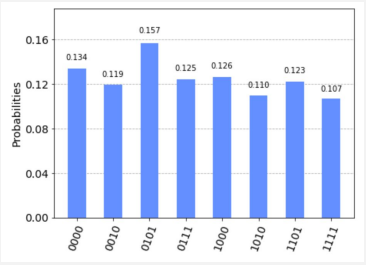


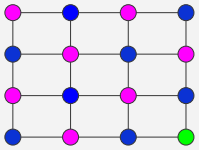
Searching quantumly

- What's the probability of measuring the target?



Walk on a Grid (Torus)

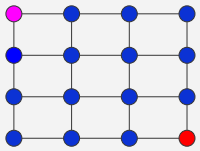




Pretty bad....

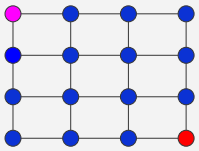
Searching quantumly: downward walk

- New idea: find a walk that covers a significant portion of the graph.



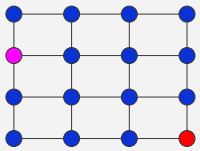
Searching quantumly: downward walk

- How about just moving downwards?



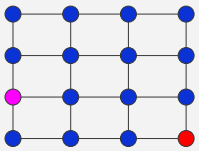
Searching quantumly: downward walk

- How about just moving downwards?



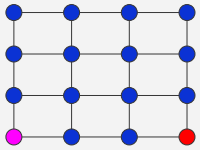
Searching quantumly: downward walk

- How about just moving downwards?



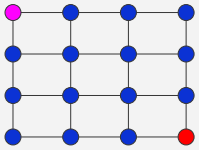
Searching quantumly: downward walk

- How about just moving downwards?



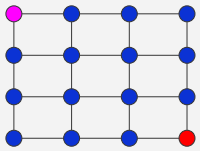
Searching quantumly: downward walk

- How about just moving downwards?



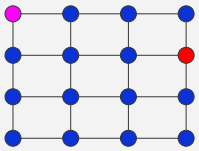
Searching quantumly: downward walk

- This covers exactly 1/n of the graph (given an n*n grid)
- Takes n steps



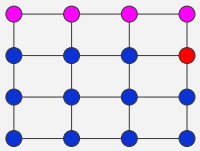
Searching quantumly: downward walk

- Idea: use superposition to cover more area



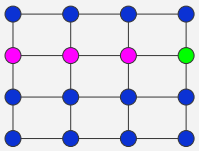
Searching quantumly: downward walk

- Idea: use superposition to cover more area



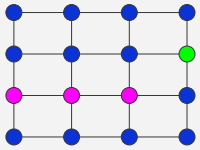
Searching quantumly: downward walk

- Idea: use superposition to cover more area



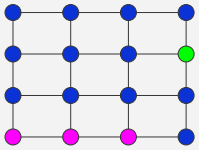
Searching quantumly: downward walk

- Idea: use superposition to cover more area



Searching quantumly: downward walk

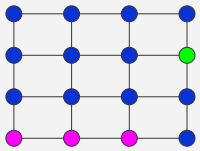
- Idea: use superposition to cover more area



Searching quantumly: downward walk

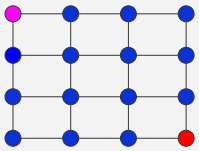
- In n steps, we are guaranteed to measure the target with probability 1/n.
- Better!

But for large values of n, it's worse…



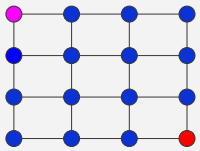
Searching quantumly: stair walk

- New idea: find a walk that covers a **large** portion of the graph.



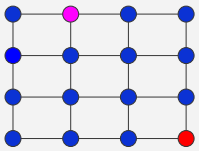
Searching quantumly: stair walk

- A simple one is "stair walk".



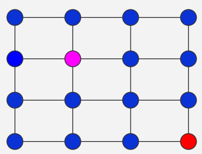
Searching quantumly: stair walk

- A simple one is "stair walk".



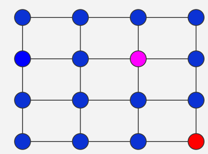
Searching quantumly: stair walk

- A simple one is "stair walk".



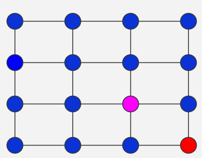
Searching quantumly: stair walk

- A simple one is "stair walk".



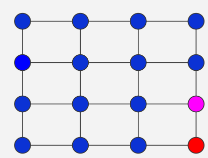
Searching quantumly: stair walk

- A simple one is "stair walk".



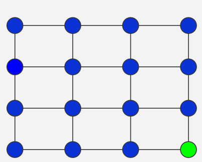
Searching quantumly: stair walk

- A simple one is "stair walk".



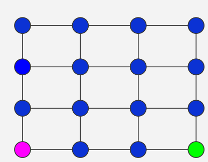
Searching quantumly: stair walk

- A simple one is "stair walk".



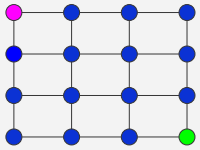
Searching quantumly: stair walk

- A simple one is "stair walk".



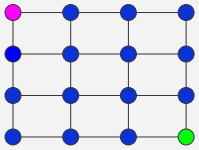
Searching quantumly: stair walk

- A simple one is "stair walk".



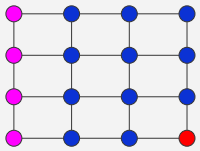
Searching quantumly: stair walk

- A simple one is "stair walk".
  - Stair walk covers exactly 50% of the graph with N/2 steps. (N = n*n = number of elements)



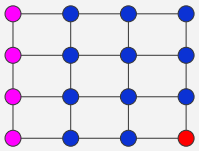
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



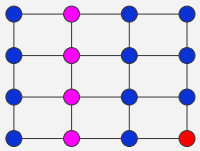
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



Searching quantumly: stair walk

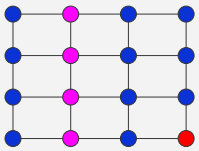
- Idea: use a superposition of states to cover more ground



Searching quantumly: stair walk

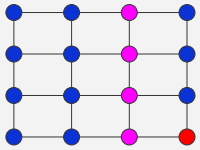
- Idea: use a superposition of states to cover more ground

All states move downward, but resulting superposition is the same!



Searching quantumly: stair walk

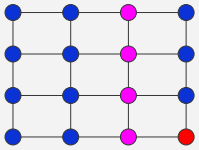
- Idea: use a superposition of states to cover more ground



Searching quantumly: stair walk

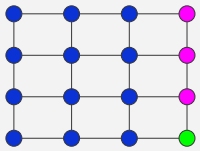
- Idea: use a superposition of states to cover more ground

All states move downward, but resulting superposition is the same!



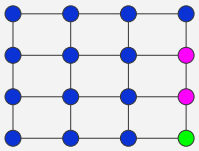
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



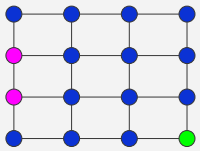
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



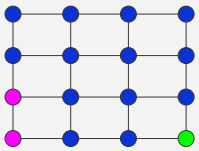
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



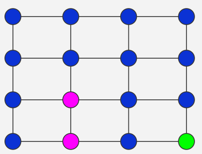
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



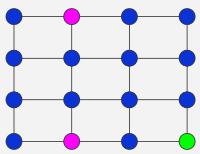
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



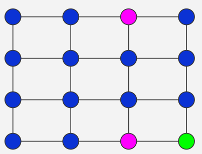
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



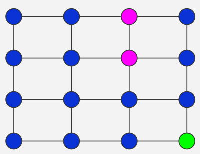
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



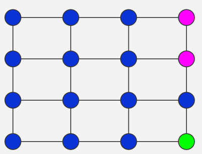
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



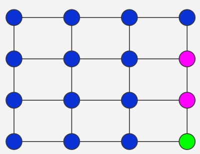
Searching quantumly: stair walk

- Idea: use a superposition of states to cover more ground



Searching quantumly: stair walk

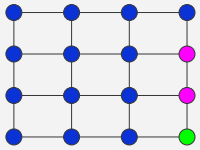
- Idea: use a superposition of states to cover more ground



Searching quantumly: stair walk

- There is a 50% chance of measuring the target after N/2 steps!
  - unfortunately, this is actually no better than classical... :(

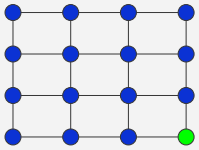
However, you might not have to take all N/2 steps depending on the size and dimensions of the graph...



if you have ideas, join me in my research!

Quantum Walk with Grovers

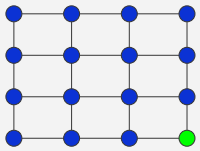
- A new idea: walking on edges



Szedgedy Quantum Walks

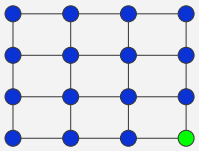
- Szedgedy walks are walks on a graph's edges
- A double-cover graph is constructed from the original graph

Grover's amplitude amplifcation process is used to select the edges adjacent to target vertices



Coined quantum walk search

- However, Szedgedy walks are equivalent to coined walks
- We will construct our own quantum walk search



Coined quantum walk search

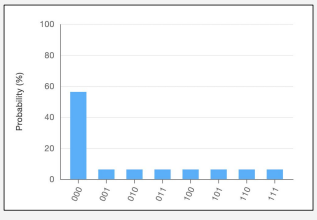
- First, we need a new coin
- Previous coin was Hadamard coin
- Now we will use "Grover's coin"

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} \frac{2}{n}-1 & \frac{2}{n} & \cdots & \frac{2}{n} \\ \frac{2}{n} & \frac{2}{n}-1 & \cdots & \frac{2}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{n} & \frac{2}{n} & \cdots & \frac{2}{n}-1 \end{bmatrix}$$

Coined quantum walk search

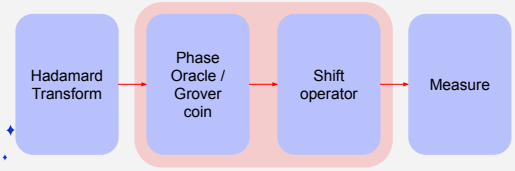
- Grover's diffuser is the coin
- It puts states into superposition similar to the Hadamard coin, but not equally



Coined quantum walk search

- How do we apply the walk?

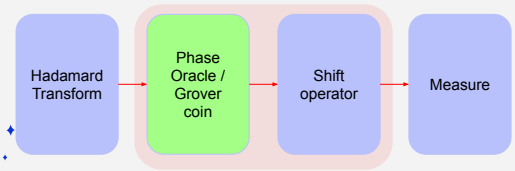
$U = SC$



Coined quantum walk search

- We must modify the coin operation
- Given state $|d\rangle \otimes |pos\rangle$, apply the following to $|d\rangle$:
  - $-I$, if $|pos\rangle$ is a target state
  - Grover's coin, if $|pos\rangle$ is a non-target state

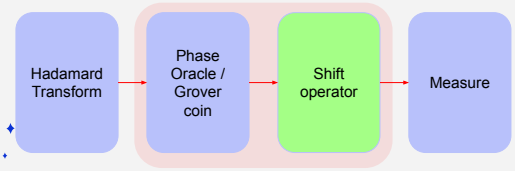
$U = SC$



Coined quantum walk search

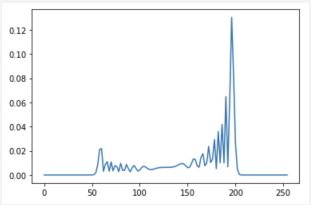
- We must also modify the shift operation
- We will only look at hypercube search spaces
- Given state $|d\rangle \otimes |pos\rangle$, flip the d^{th} bit in the pos bitstring

$U = SC$



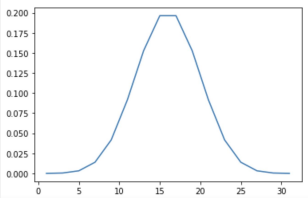
Noise distributions

- Let's take a look at the Hadamard distribution.



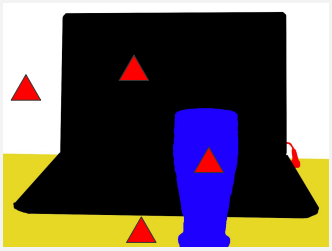
Noise distributions

- We can also create the Gaussian distribution!



Other (Possible) Applications

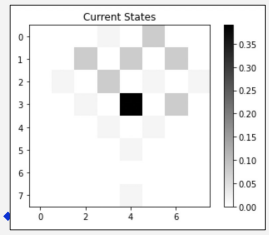
Image segmentation



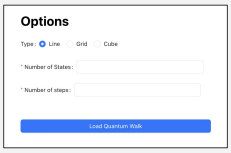
- We can let all pixels walk at once!

My Research

Teaching other people about quantum walks!



- Visualization software

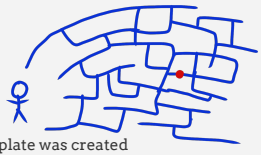


Demo

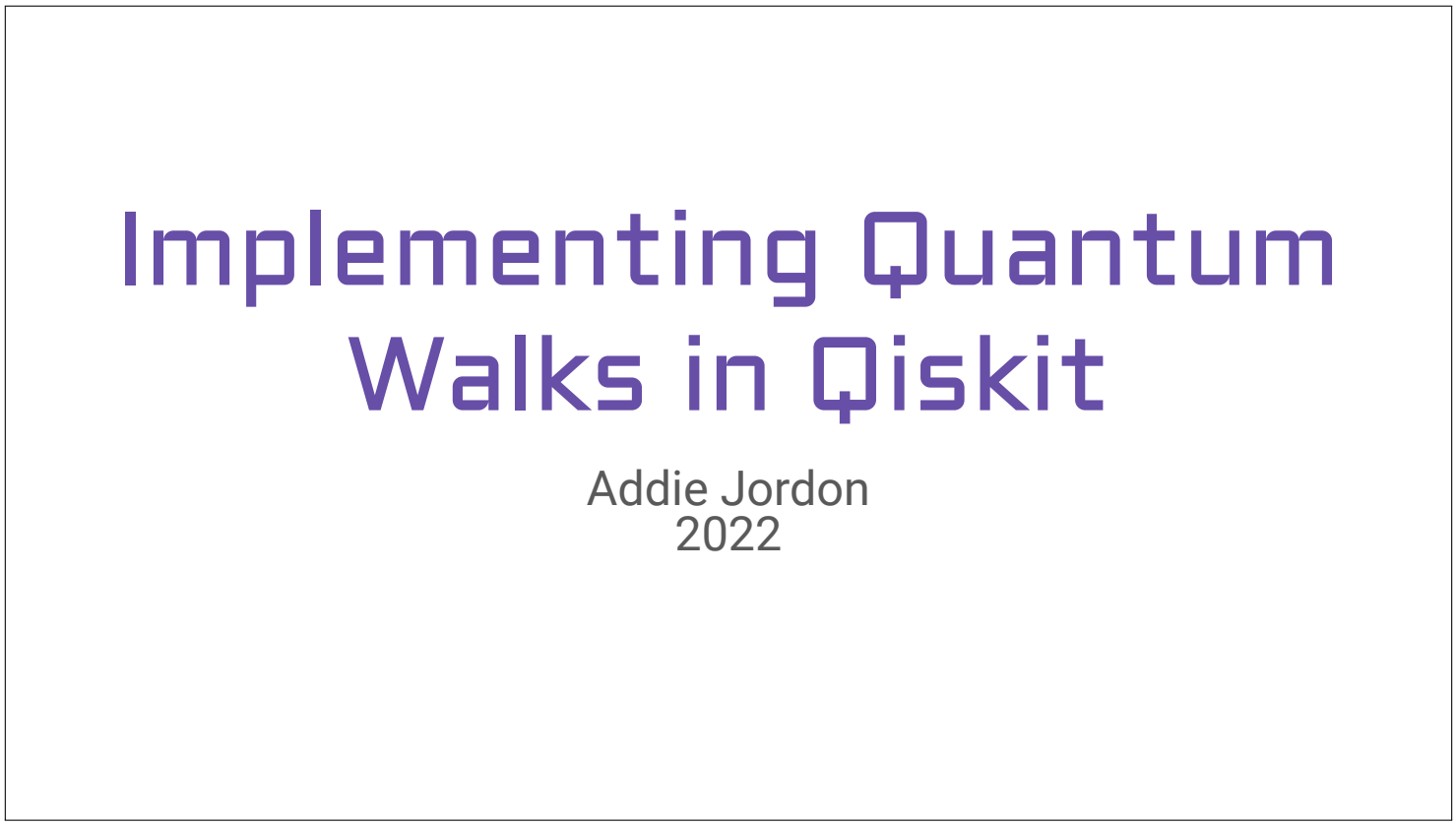
THANKS!

Questions? Ideas? Feedback?

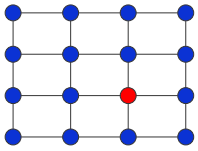
addie@uvic.ca



B Sample Lecture Slides for Implementing Quantum Walks

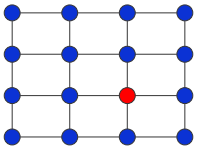


Required Components



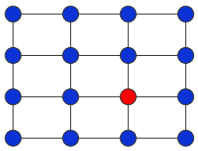
- an oracle
  - we will use a Boolean oracle (ie. ancilla)
- the Shift operator (S)
  - an incrementer / decrementer
- the Coin operator (C)

Shift operator: moving right



- top row
  - 0000 -> 0001,
  - 0001 -> 0010
  - 0010 -> 0011
  - 0011 -> 0000
- second row
  - 0100 -> 0101
  - 0101 -> 0110
  - 0110 -> 0111
  - 0111 -> 0100
- etc

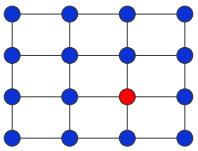
Shift operator: moving right



- top row
  - 0000 -> 0001,
  - 0001 -> 0010
  - 0010 -> 0011
  - 0011 -> 0000
- second row
  - 0100 -> 0101
  - 0101 -> 0110
  - 0110 -> 0111
  - 0111 -> 0100
- etc

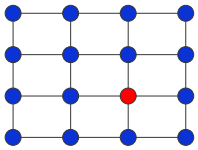
On a 4x4 2-dimensional lattice, moving right amounts to incrementing the rightmost 2 qubits!

Shift operator: moving left



- top row
  - 0000 -> 0011
  - 0001 -> 0000
  - 0010 -> 0001
  - 0011 -> 0010
- second row
  - 0100 -> 0111
  - 0101 -> 0100
  - 0110 -> 0101
  - 0111 -> 0110
- etc

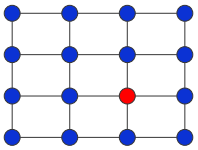
Shift operator: moving left



- top row
  - 0000 -> 0011
  - 0001 -> 0000
  - 0010 -> 0001
  - 0011 -> 0010
- second row
  - 0100 -> 0111
  - 0101 -> 0100
  - 0110 -> 0101
  - 0111 -> 0110
- etc

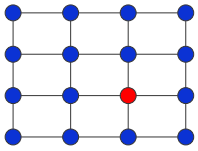
On a 4x4 2-dimensional lattice, moving left amounts to decrementing the rightmost 2 qubits!

Shift operator: moving up



- left column
  - 0000 -> 1100
  - 0100 -> 0000
  - 1000 -> 0100
  - 1100 -> 1000
- second column
  - 0001-> 1101
  - 0101 -> 0001
  - 1001 -> 0101
  - 1101 -> 1001
- etc

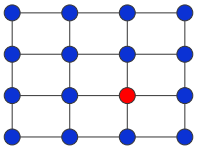
Shift operator: moving up



- left column
  - 0000 -> 1100
  - 0100 -> 0000
  - 1000 -> 0100
  - 1100 -> 1000
- second column
  - 0001-> 1101
  - 0101 -> 0001
  - 1001 -> 0101
  - 1101 -> 1001
- etc

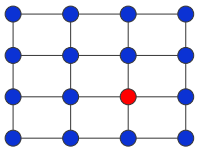
Moving up amounts to decrementing the leftmost 2 qubits!

Shift operator: moving down



- left column
  - 0000 -> 0100
  - 0100 -> 1000
  - 1000 -> 1100
  - 1100 -> 0000
- second column
  - 0001-> 0101
  - 0101 -> 1001
  - 1001 -> 1101
  - 1101 -> 0001
- etc

Shift operator: moving down

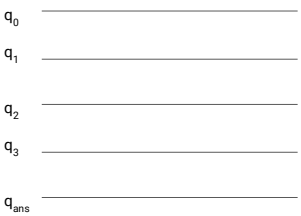


- left column
  - 0000 -> 0100
  - 0100 -> 1000
  - 1000 -> 1100
  - 1100 -> 0000
- second column
  - 0001-> 0101
  - 0101 -> 1001
  - 1001 -> 1101
  - 1101 -> 0001
- etc

Moving down amounts to incrementing the leftmost 2 qubits!

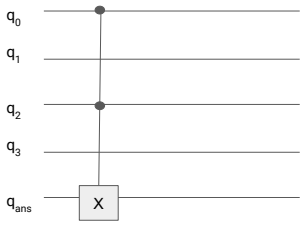
Write the Shift operator in Qiskit

Oracle



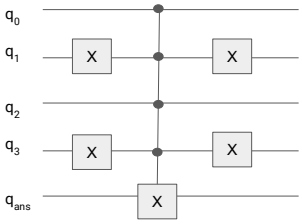
- We will use an ancilla qubit to mark the target state
  - note: for this walk, we will not change the phase of the target
  - thus, we will not use Phase Kickback

Oracle



- Let's say the target is 1010
  - How do we make sure only 1010 is marked?
- We might be tempted to do this....
  - but this would mark also 1111, 1110, and 1011

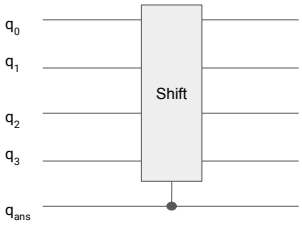
Oracle



- Let's say the target is 1010
  - How do we make sure only 1010 is marked?
- Idea: make sure the 0's are 0's by adding X gates!
  - control on all position qubits

Write the Oracle in Qiskit

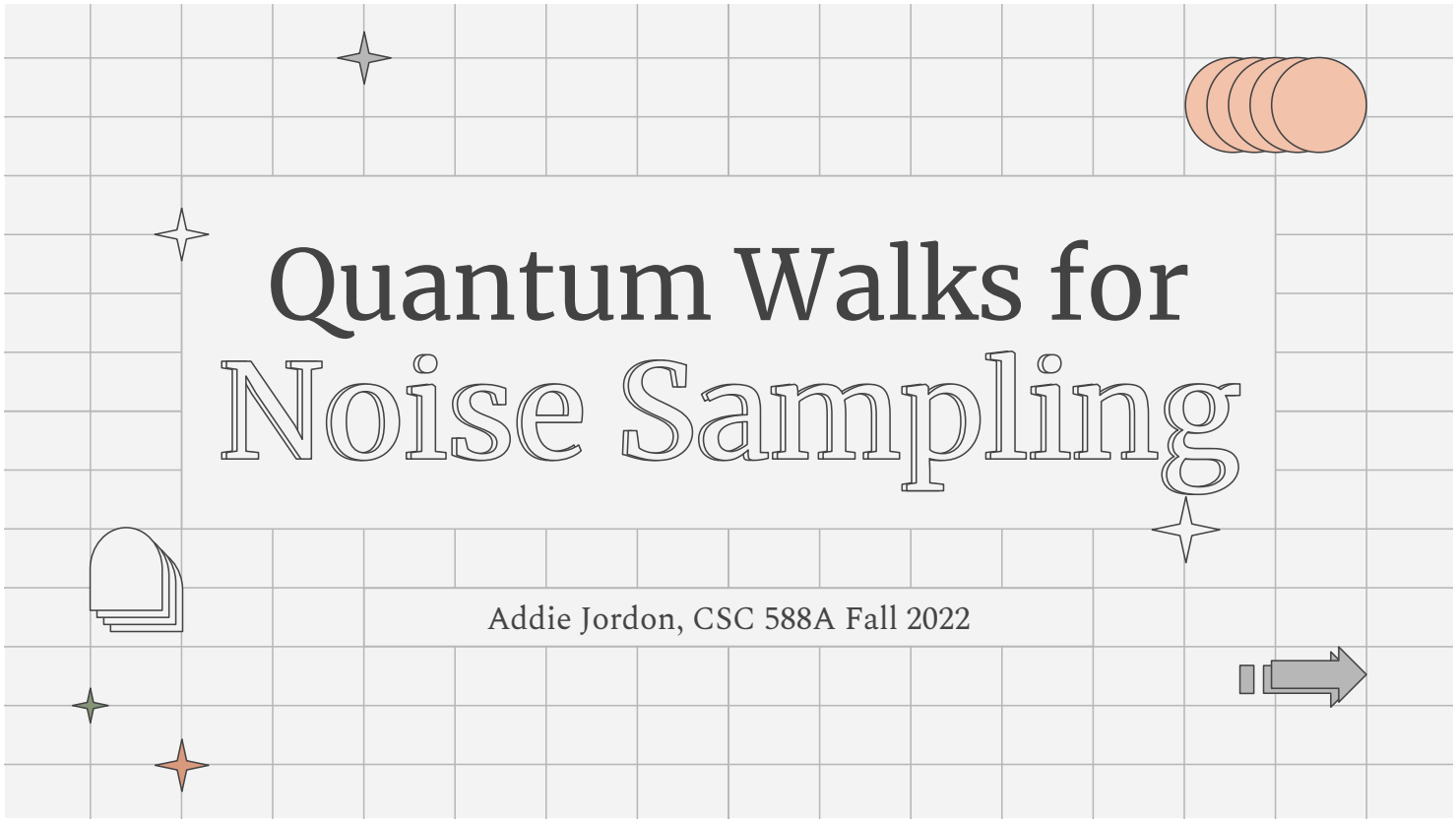
Applying the shift



- We will only shift if we are not in the target state
  - we will use the ancilla to control the Shift operator

Put it all together in Qiskit

C Sample Lecture Slides for Quantum Walks for Noise Sampling

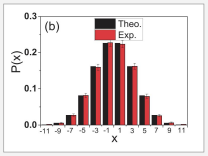


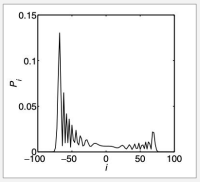
01

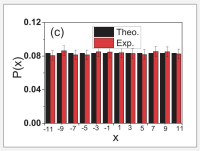
Introduction

Using quantum walks to create noise distributions

Idea: use quantum walks to create probability distributions





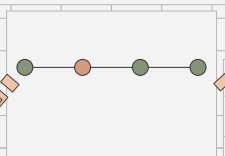


Mackay TD, Bartlett SD, Stephenson LT, Sanders BC. Quantum walks in higher dimensions. Journal of Physics A: Mathematical and General. 2002 Mar 15;35(12):2745.

R. Zhang, R. Yang, J. Guo, C.-W. Sun, Y.-C. Liu, H. Zhou, P. Xu, Z. Xie, Y.-X. Gong, and S.-N. Zhu, "Arbitrary coherent distributions in a programmable quantum walk," Physical Review Research, vol. 4, no. 2, p.023042, 2022

Review: Quantum Walks

Classical random walks



Quantum walks

Project Proposal

Implementation

I will implement quantum walks on a line and possibly on a lattice in Python using Qiskit.

Noise Distributions

I will look at creating the Gaussian mechanism and something akin to the exponential mechanism.

Timeline

Given that my research is in implementing quantum walks, I have existing code which I can modify to serve for noise sampling.

Questions to answer...

- Which probability distributions can I create using QWs?
- What are the challenges in creating probability distributions quantumly?
- Is it easier to create symmetric or asymmetric distributions?
- Can I create intuitive graphics to display my findings?

02

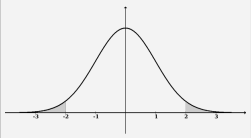
Motivation

Quantum as a source of randomness

Additive Mechanisms

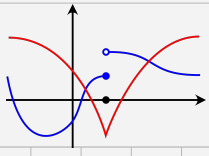
Gaussian (Normal)

- $f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi * \sigma^2}} * e^{-\frac{1}{2} * (\frac{x-\mu}{\sigma})^2}$



Exponential

- $\exp(\varepsilon \bullet u(D,r)/2\Delta u)$



What do they have in common?

1. Probability

With some probability, a noisy number is chosen.

This requires...

2. Randomness!

Even with a probability distribution, the end result (ie. picking of the number) **should** be completely random.

True randomness

Is it even needed?

For the most part, pseudo-randomness is sufficient.

01

Cryptology (+ Privacy)

Since both have high stakes, the requirement for truly random numbers is higher.

02

Additive mechanisms

Differential privacy needs truly random numbers as long as it uses additive mechanisms!

S. L. Garfinkel and P. Leclerc, "Randomness concerns when deploying differential privacy," in Proceedings of the 19th Workshop on Privacy in the Electronic Society, 2020, pp. 73–86.

Bosley C, Dodis Y. Does privacy require true randomness?. InTheory of Cryptography Conference 2007 Feb 21 (pp. 1-20). Springer, Berlin, Heidelberg.

Sources of randomness?

Sources of true randomness are extremely difficult to find.

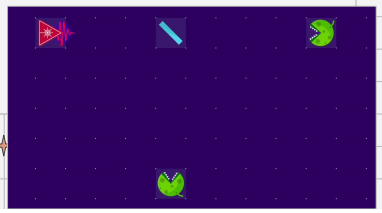
- throwing dice
- picking objects from a hat
- shuffling playing cards

All resulted in biases such as an unusual number of 5s and 6s.

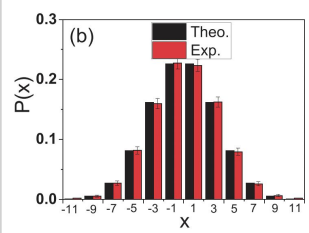
What can we use for randomness????

Quantum is random

The idea behind quantum computing is that quantum bits can be both 0 and 1 at the same time...



Idea: use quantum walks to create probability distributions



Mackay TD, Bartlett SD, Stephenson LT, Sanders BC. Quantum walks in higher dimensions. Journal of Physics A: Mathematical and General. 2002 Mar 15;35(12):2745.

R. Zhang, R. Yang, J. Guo, C.-W. Sun, Y.-C. Liu, H. Zhou, P. Xu, Z. Xie, Y.-X. Gong, and S.-N. Zhu, "Arbitrary coherent distributions in a programmable quantum walk," Physical Review Research, vol. 4, no. 2, p.023042, 2022

03

Tasks

Completed and to-do

Tasks

Hadamard walk

Take the Hadamard walk and plot its distribution

Tasks

Hadamard walk

Take the Hadamard walk and plot its distribution

Gaussian / Normal dist

Force Hadamard walk into Gaussian distribution

Tasks

Hadamard walk

Take the Hadamard walk and plot its distribution

Gaussian / Normal dist

Force Hadamard walk into Gaussian distribution

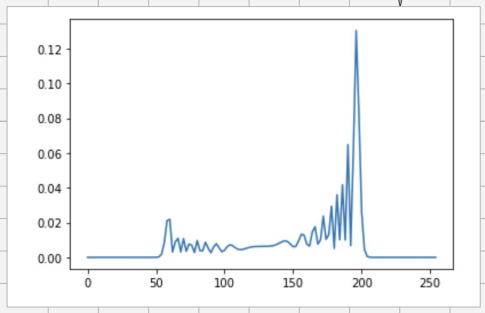
Equal dist*

Plot equal distribution

Tasks

Hadamard walk

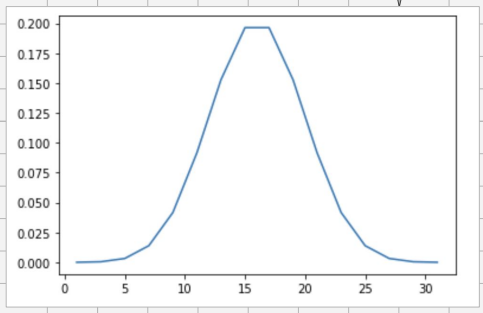
Take the Hadamard walk and plot its distribution



Tasks

Gaussian / Normal dist

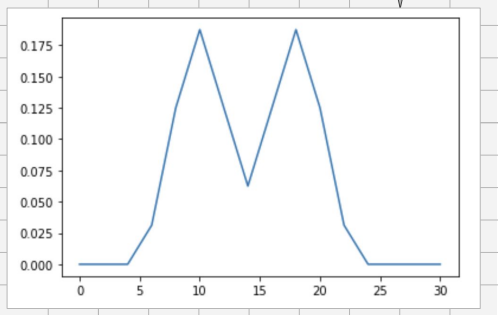
Force Hadamard walk into Gaussian distribution



Tasks

Gaussian / Normal dist

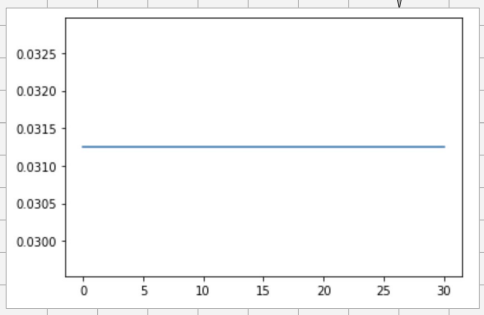
Bonus: Add multiple walkers to simulate custom distribution



Tasks

Equal dist*

Plot the equal distribution



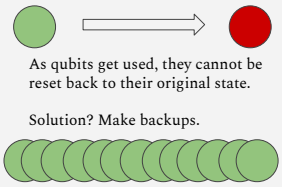
04 Challenges

Technical issues

Quantum development

Ancilla qubits

- in order to get the Gaussian distribution, direction has to be "reset"
- actual "resetting" of qubits breaks unitary properties of quantum mechanics, so ancilla qubits were used to avoid this



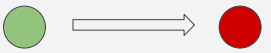
As qubits get used, they cannot be reset back to their original state.

Solution? Make backups.

Quantum development

Ancilla qubits

- in order to get the Gaussian distribution, direction has to be "reset"
- actual "resetting" of qubits breaks unitary properties of quantum mechanics, so ancilla qubits were used to avoid this



As qubits get used, they cannot be reset back to their original state.

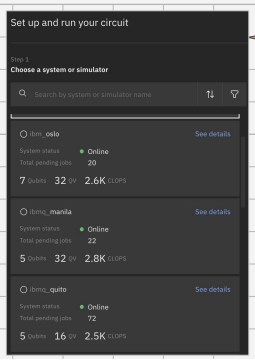
Solution? Make backups.

Problem: now number of qubits required (size complexity) scales linearly with number of iterations.

Quantum computers

Quantum computers are... bad.

- limited number of qubits
- choice between simulator and real quantum computer
- simulator is just a classical computer → long time
- quantum computer offers no insights for research
- also quite noisy, limited qubits, or expensive

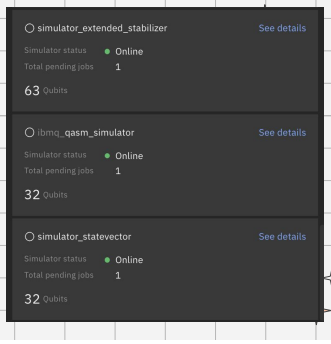


Quantum computers

Quantum computers are... bad.

- limited number of qubits
- choice between simulator and real quantum computer

- simulator is just a classical computer → long time
- quantum computer offers no insights for research
- also quite noisy, limited qubits, or expensive



Quantum computers

Quantum computers are... bad.

- limited number of qubits
- choice between simulator and real quantum computer

- simulator is just a classical computer → long time
- quantum computer offers no insights for research
- also quite noisy, limited qubits, or expensive

Hadamard Walk, 8 qubits, 100 steps

```
Time for quantum walk to run (s): 70.869145154953
Time for statevectors(s) to be produced (s): 29.460327863693237
Total time (s): 100.32947301864624
```

Normal / Gaussian, 5 qubits, 15 steps

```
Time for quantum walk to run (s): 1.5705878734588623
Time for statevectors(s) to be produced (s): 78.75355911254883
Total time (s): 80.32414698600769
```

Normal / Gaussian 2 walkers, 5 qubits, 15 steps

```
Time for quantum walk to run (s): 1.1304597854614258
Time for statevectors(s) to be produced (s): 80.11094808578491
Total time (s): 81.24140787124634
```

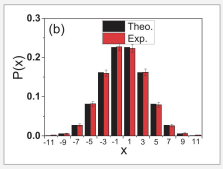
Parity problem

Walk positions

- either all even or all odd

- easiest solution → only use either an even or odd number of steps
- downside: the number of qubits used to represent position is not efficient

This can make mapping the domain difficult.



05 Demo

Jupyter notebook

Thank you!

Questions?

D Links to Quantum Walk Github Repositories

- `https://github.com/addie43110/qwalk-visualizations` contains Jupyter notebooks for rendering two- and three-dimensional visualizations for Quantum Walks.
- `https://github.com/addie43110/qwalk-app` contains the code for the front end of the QWalkVis website application.
- `https://github.com/addie43110/qwalk-app-backend` contains the code for the back end of the QWalkVis website application.
- `https://github.com/addie43110/qw-noise-sampling/blob/main/Quantum%20Walks%20Distributions%20for%20Noise%20Sampling.ipynb` contains a Jupyter notebook for creating distributions for noise sampling using quantum walks.